Crowdsource Requirements Engineering: Using Online Reviews as Input to Software Features Clustering

Noor Hasrina Bakar[1], Zarinah M. Kasirun[2], Norsaremah Salleh[3] and Azni H. Halim[4]
[1]*Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia.*
[2]*Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia.*
[3]*Kuliyyah of Information and Communication Technology,*
International Islamic University Malaysia, 50728 Kuala Lumpur, Malaysia.
[4]*Faculty Science and Technology, Universiti Sains Islam Malaysia, 71800 Nilai, Negeri Sembilan, Malaysia.*
noorhasrina@ukm.edu.my

***Abstract*—As to date, various software being produced to help in our daily routines. At times, there are complaints on errors or faults lodged by users over the internet. This information can be valuable for software development teams to enhance the software functionalities in the next releases. Not only that, these comments contain important software features that can be extracted and reuse for future development of similar software systems. Reviews provided by various user from unknown background is an example of open call involvement in crowdsource software engineering. In this paper, sample software reviews available in the internet were collected. In the experiment conducted, twenty-five groups of random software reviews within the domain of children online learning software were selected as input to crowdsource requirements engineering. The extracted reviews were then clustered into related groups by using K-Means algorithm. The clustering results achieved by K-Means were evaluated in terms of cluster compactness and cohesion. A statistically significant result with time efficiency obtained and reported at the end of this paper. Based on this information, this paper provides some recommendations on how user reviews can be used as input to the crowdsource requirements engineering either for improving existing software or for production of a new similar systems.**

***Index Terms*—Crowdsource Software Development; Feature Extraction; Requirements Engineering; Similar Systems Development.**

I. INTRODUCTION

It is a norm for users to leave comments after they have experienced certain software, especially when they are unsatisfied with it. For example, users downloaded an online hotel booking apps. After encountering some difficulties in certain function, frustrated users may leave some comments about the problems, with the hope somebody from the developer's side will come and fix it. Developers have taken user comments to come up with a similar software for a better version: the same development team fixed the problems and published a newer improved version, or a different development team (business competitor) uses the comments to come out with a similar software. with an enhanced version. Here, the feedbacks on software functionalities are contributed by users in open call format; the concept of crowdsourced software engineering.

Crowdsource software engineering emphasizes any software engineering activity, involving any act of undertaking any external software engineering tasks by an undefined, potentially large group of online workers in open call format [1]. Meanwhile, the use of user review as a source for gathering software requirements received attention from software engineering community and this can be seen in [2]. As an example, a detailed systematic review on mining user reviews from mobile applications was published in [3]. The review mechanism provided for software and mobile apps resembles a communication channel that bridge the gap between users and developers[1], at the same time supporting the crowdsource software development.

The act of open user involvement in terms of providing feedback for improvement may benefit the development team and other users in future. However, when massive comments are received, developers will face hard times in digesting information from reviews if done manually. In order to extract related software functionalities, there is a need for a simple automated process to classify the user comments. For example, positive comments from users can be a channel to boost team motivation, while negative comments must be taken seriously for future improvements. This research is interested to extract open comments related to software features and highlight this to the development team.

This paper firstly describes related works in the area of crowdsource software development and requirements extraction from user reviews. In Section III, the proposal for a crowdsource requirements engineering process is presented. Then, the results from an initial experiment on crowdsource requirements engineering is discussed in Section IV, and lastly this paper is concluded by highlighting thread to validity and the future plans for this research.

II. RELATED WORK

In order to describe the characteristics of crowdsource based software development, let us first take a closer look at the traditional requirements elicitation process. In practice, requirements elicitation is a process of gathering the requirements of a system from users, customers and stakeholders. Requirements elicitation usually involves development teams going back and forth discussing with stakeholders until requirements are mutually agreed. There, stakeholders and development team discussed in closed and

formal meetings. The input to software development, the software functionalities must be agreed upon by all stakeholders, and this information remain secrecy until the first version of the software product is released. This is different from crowdsource software development where requirements or input to the development of similar system or new releases are based on crowd's opinion in an open call. The term crowdsourcing was firstly coined in [5], and crowdsourcing is explicitly defined as any acts a company or institution taking a function once performed by employees, and outsourcing it to an undefined (and generally large) network in the form of an open call. Open call here means anybody interested may contribute to the development, and this is made possible with the existence of internet. In this section, we divide our review into two distinct areas: related works on crowdsource software development and related works on requirements extractions from user reviews. From this point on, we will refer the crowdsource software engineering as Crowd SE.

Within software engineering community, [1] surveyed 210 unique publications on Crowd SE practices. Selected publications ranged from 2008 – 2015. They have summarized the following items: Crowd SE practice in terms of platforms and case studies available, Crowd SE Applications in terms of published researched in the area of Crowdsourcing Requirements Analysis, Software Design, Coding, Testing and Verification, Evolution and Maintenance and crowdsource for other software engineering activities. Some benefits of Crowd SE from case studies been discussed in [6] and [7]. Stol and Fitgerald reported a case study on implementation of Crowd SE in a multi-national corporation, TechPlatform Inc. (known as TPI), a global player that offers services and solution in the cloud [7]. TPI outsourced its software development through TopCoder, the largest software development crowdsourcing platform with more than 600,000 developers who codes software for clients. Through TopCoders, softwares are produced based on competition held online. Developers compete for prizes to produce best softwares. Winning software will be licensed for profit by TopCoder. Hitherto, important organization like Facebook, Amazon, NASA, Google and more leverages on this competition based program development to select the most innovative software produced by the most brilliant competitors. Since timely delivery of the software is very crucial, as long as deadline is met, Crowd SE provides flexible working schedule for developers[6]. This is due to its nature where development can take place anytime anywhere as long as there is internet connection, developers can freely choose their working place and schedule.

Although Crowd SE provide added value in terms of faster software delivery and flexible development schedule, the following are some prevalent issues and problems discussed in [1]: quality control mechanism, difficult task decomposition for complex software, planning, scheduling, motivations and remunerations are among issues related to Crowd SE. Even if task can be properly decomposed, getting the correct specification still remain as global issues in Crowd SE [8], with the question: can requirements be crowdsourced? Software engineering research community have begun to study this problem including how to create workflow for a variety of development tasks for Crowd SE. For example, StakeSource 2.0, a tool that identifies and prioritises stakeholders and their requirements by using social networks[8]. Among the features offered by StakeSource 2.0 includes the collection of requirements and their ratings, recommendations of other requirements of interest and visualization of requirements on the social networks. Communication between analysts and stakeholders occurs through emails and social network site. Ratings are given and requirements are prioritized based on overall ratings from stakeholders. StakeSource 2.0 highlights any stakeholders' conflicts pertaining to specific requirements, and reveal this issue on the social network. Consequently, attention should be given to requirements with many stakeholders in conflict. StakeRare is another example of Crowd SE for requirements elicitation which uses social network and collaborative filtering to identify and prioritize requirements in large software projects [10]. StakeRare identifies stakeholders for a project and ask stakeholders involved to recommend other stakeholders and build a social networks of stakeholders for a posted project. Links for their recommendations are shared and stakeholders are asked to rate initial requirements lists, and the system recommends other relevant requirements using collaborative filtering. Other proposal to enhance the use of Crowd SE in requirements engineering can as well be found in CrowdREquire[11], UDesignIt, Bespoke and AOI[1].

Up to this section, the related work in Crowd SE is briefly described and followed by how Crowd SE being applied in requirements engineering activities. Next, let us take a look at brief overview on the use of user reviews as input to requirements engineering process. Guzman and Maalej [2] used collocation findings to extract fine-grained features, utilised sentiment analysis to extract sentiments and opinions associated to the features, and applied topic modelling to group-related features. They have extracted 32210 reviews for 7 iOS and Android apps and compared the results with 2800 manually peer-analysed reviews. The results indicate that their proposed approach is effective in extracting the most frequently mentioned features. Groups of features are coherent and relevant to app requirements, and sentiment analysis results positively correlate to the manually assigned scores. In their work [2], the extraction process was done by using the NLTK toolkit. Nouns, verbs, and adjectives were extracted from the mobile app reviews, followed by stop words removal. This was then followed by lemmatization process that grouped different inflected words with the same part of speech tagging together (lemmatization group words that are syntactically different but semantically similar). The collocation algorithm provided by the NLTK toolkit was then applied for extracting features from the reviews.

Carreno and Windbladh [4] analysed the user reviews available for third-party mobile applications as a way to extract new or changed requirements for future releases of a particular software. In their work, the authors used topic modelling to extract the main topics from the user feedback and evaluated them on different publicly available data sets.

In previous work, we have explored the use of software reviews as input to feature extraction from natural language, FENL [12] to assist the reuse of requirements, by using natural language processing and information retrieval techniques. There, a promising precision result was obtained when compared to manual process. In this paper, we will apply the FENL approach and extend its functionality by incorporating k-Means clustering algorithms. Based on the experiment conducted, we will provide suggestion on how this can be applied to crowdsource requirements engineering.

III. OUR APPROACH

The main goal of our approach is to automatically identify and cluster the software functionality from the selected software reviews published by the crowd. This information will then be highlighted to development team for further actions. For this, open source screenscraper tool is used to copy the user comments from the internet. These raw data are then stored in text files and fed into the FENL tool. The k-Means clustering is used to cluster the similar software features together. Lastly, these features will be highlighted to development team for further actions. Figure 1 illustrates the steps in our approach.

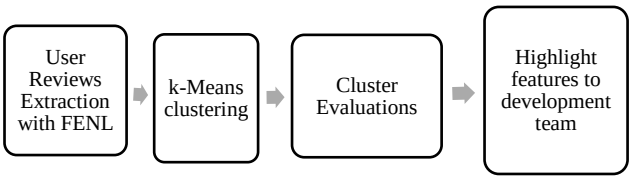


Figure 1: Crowdsource RE proposal

In the following sub-section, we describe the first two processes from Figure 1 (User Review Extraction with FENL and k-Means clustering). The later two processes: "Cluster Evaluations" and "Highlight Features to Development Team" will be discussed in Section IV.

A. Review Extraction

In this work, a total of 25 software reviews pertaining to online children learning posted at toptenreviews.com are scraped, by using open source screen scraper utility. This 25 software reviews came from three subcategories under the domain of online learning software for children: Preschool games (10 reviews), Algebra (7 reviews) and Creative Writing (10 reviews). Complete description of Feature Extraction from Natural Language, FENL can be found in [11]. Here, the FENL tool is used to extract the features that resides from the 25 reviews. As the output, FENL produces list of features in the forms of noun phrases.

Figure 2 illustrates some example the example of noun phrases extracted:

```
'math recommend', 'plan take', 'may want', 'menu pop', 'option guide', '
 take', 'phone receive', 'manner help', 'page take', 'need solve', 'progr
ıd', 'job explain', 'need take', 'tool solve', 'possibility learn', 'endl
ıs', 'program teach', 'software combine', 'tutorial exercise', 'look impr
ɔlace start', 'advantage cover', 'would find', 'advantage cover', 'subjec
ɨe', 'knowledge make', 'refresh learn', 'people decide', 'program take',
ɛt', 'concept feel', 'advantage help', 'math beginning', 'function jump',
ıelp', 'subject divide', 'example perform', 'advantage explain', 'good di
ɛed', 'variety find', 'program learn', 'program come', 'grade improve', '
ɔrtcoming', 'program help', 'algebra improve', 'subject cover', 'advantag
ɨ', 'value advance', 'confidence learn', 'study cover', 'may take', 'prin
ɔlve', 'software start', 'beginning get', 'section want', 'section know',
```

Figure 2: Sample noun phrases extracted from software reviews

To obtain phrase relatedness, each noun phrases is being tabulated in the term-document matrix, in terms of the number of its occurrences. This is followed by applying the term-frequency-inverse-document-frequency, tf-idf. We applied the Singular Value Decomposition, SVD calculation from the Latent Semantic Analysis, LSA. The LSA application output the coordinates for all the phrases in the document space. The closer the distance between a noun phrase to another indicate that they are of similar meaning, with the assumption of similar terms tend to occur in similar contexts [13].

B. K-Means Clustering

K-Means clustering is used to group the noun phrase (feature summary) together. The K-Means algorithm is the simplest and commonly used algorithm to optimize the objective function (the distance) that is described by the following equation:

$$E = \sum_{i=1}^{c} \sum_{x \in C_i} d(x, m_i) \quad (1)$$

where mi is the centre of cluster Ci, while d(x,mi) is the Euclidean distance between data point, x and mi. In the experiment, we input the coordinates obtained from LSA and we then use K-means algorithm (based on [14]) indicated as follows:

i. Specify a fixed number of clusters, c.
ii. Randomly pick up a cluster centre.
iii. Assign all data points (the coordinate for terms obtained earlier) to the cluster whose centre is the nearest (closest centroid).
iv. Re-compute the centres for each centroid.
v. Repeat the process in steps iii & iv until the centres stop changing.

The clustering experiment was conducted against the extracted noun phrases to form c number of clusters, for three different rounds at random numbers, c=8,10, 12, and 15); and each round of clustering was executed within 100 iterations.

IV. RESULTS

There were 391 noun phrases extracted: 173, 107 and 111 distinct phrases from the three software reviews, as shown in Table 1:

Table 1
Dataset for the experiment

Learning Software Subcategory	# of documents	Noun phrases
Preschool Learning	10	173
Algebra Learning	6	107
Writing Software	9	111
Total	25	391

Noun phrases extracted can be used to demonstrate the main theme or main features from the software reviews. For example, among noun phrases extracted include "preschool games", "coloring pages" and "algebra software", which provide us the general ideas on the basic features for the software being reviews.

K-Means clustered the similar features according to the distance of each noun phrases within the document space as shown in Figure 3. Note that with c=8, 19 noun phrases were grouped in one cluster, and the remaining two are separated. Logically, this is not accurate. For example, phrases related to "maths" should have been separated to phrases related to "coloring activities", but with c=12 and c=15, both of the phrases were grouped together, leading to assumption that less cluster might not produce accurate groupings.

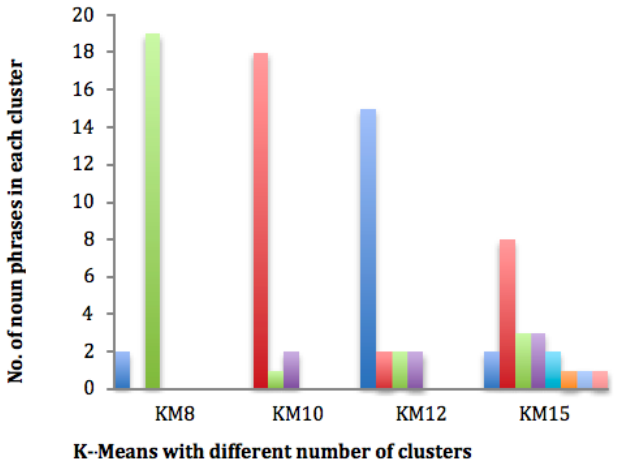


Figure 3: The k-means clustering results

Next, the cluster validation for the k-Means clustering is presented in terms of: 1) Inter-cluster distance, 2) Intra-cluster distance and statistical evaluations

Figure 4 illustrates the difference between inter-cluster and intra-cluster distance, where point X indicates the centroid for each clusters, while points labeled o are the items reside within each cluster.

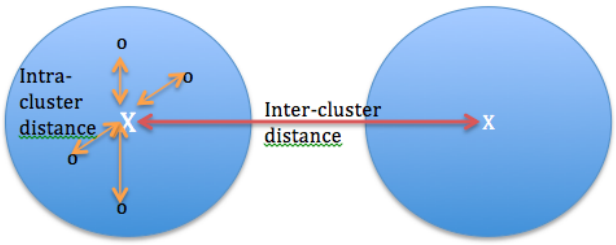


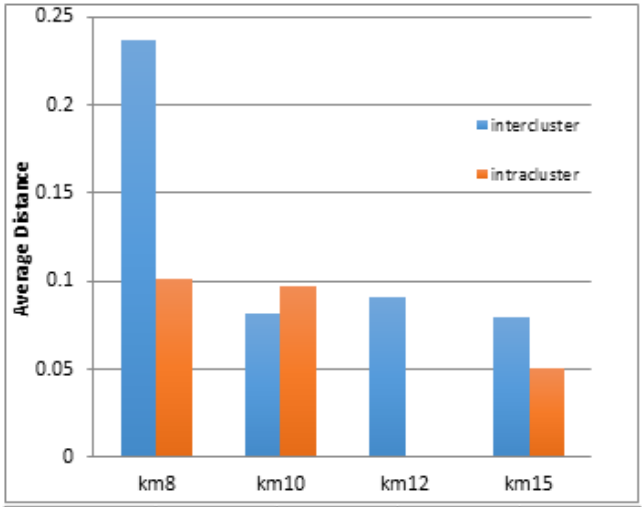
Figure 4: Intra-cluster distance and inter-cluster distances

A. Inter-cluster Distance (separation)

To obtain the inter-cluster distance, the average distance between all centroids produced by the K-Means is taken; this sometimes referred to as the degree of separation from one centroid to other centroids. This is accomplished by applying the following function in Matlab:

$$m = pdist(centroids) \quad (2)$$

where centroids are the list of cluster centers obtained by the K-Means algorithm. The bigger the average distance indicated that a cluster produced is well separated from the other clusters in the problem space. In this observation, K-Means clustering with c=8 (i.e. km8) recorded the highest average inter cluster distance, when compared to the clustering solution with different number of clusters. In this case, smaller number of clusters tend to produce clusters that are well separated, and this is expected. Figure 5 shows the average inter-cluster and intra-cluster distance produced when using different number of clusters:



intercluster	0.236286	0.081293	0.090326	0.079721
N	8	10	12	15
intracluster	0.100809	0.097215	0.000143	0.050003
N	8	45	45	105

Figure 5: Results for average inter and intra cluster distances

To determine whether there was statistically significant difference between the means produced when clustering using different number of clusters, we had conducted a One-Way ANOVA test. The results from One-way ANOVA test (Table 2) indicate that there was significant difference in between means for all groups when measuring the inter-cluster distance, (i.e. $F(3,199) = 4.792$, $p = 0.003$). The post hoc test confirms that there was a significant mean difference on inter-cluster distance between km8 and the other groups (clusters).

Table 2
ANOVA test for Inter cluster distance

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.170	3	.057	4.792	.003
Within Groups	2.357	199	012		
Total	2.527	202			

B. Intra-cluster Distance (cohesion)

The intra-cluster distance identified the degree of compactness or cohesion of each cluster produced by the clustering algorithms.

In other words, the intra-cluster distance measures how close related items that are placed in each cluster. In the context of our experiment, we are measuring the average distance between all points to its centroid. This is accomplished by applying the Matlab code in Figure 6. Smaller intra-cluster value may indicate a better clustering result, which means items clustered within each group are very similar. In this case, c=12 to produce a more compact cluster when compared to other groups.

The ANOVA result (as in Table 3), showed that at $p < 0.05$, there was a statistically significant difference between the average distance calculated by clustering solution for different number of clusters (i.e. $F(3,41)= 3.141$, $p=0.035$). Moreover, a post hoc Tukey test conducted confirmed that c=12 differed significantly in their average distance compared to the other groups.

```
%sum for mean of each cluster
Centroids=KM12;%fetch the centroid obtained from KMeans
num=km12_coord_idx;%coordinates for x,y and the cluster index
    sum = zeros(size(Centroids,1),1);
    noel = zeros(size(Centroids,1),1); %#of elements for each id

%finding the distance from each centroid
for i = 1:size(num,1)
    pt = num(i,1:2);
    id = num(i,3);
    D = pdist2(pt,Centroids(id,:),'euclidean');
    dist(i)=D; %store into an array dist on each iteration
    sum(id) = sum(id) + D;
    noel(id) = noel(id) + 1;

end

%Finding the mean distance
for j = 1:size(Centroids,1)
    if(noel(j)==0)
        avg(j) =0;
    else
        avg(j) = sum(j)/noel(j);
    end
end
```

Figure 6: Matlab code for finding average intra-cluster distance

Table 3
ANOVA for Intra-cluster distance

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.071	3	.024	3.141	.035
Within Groups	.309	41	.008		
Total	.380	44			

V. Discussions

From the experiment, we observed that FENL able to extract software features, confirming the result obtained in our earlier experiment in [11]. By incorporating k-Means clustering, related features can be grouped based on their relatedness (distance based on LSA calculation). The final clustering results (given a suitable number of clusters) can suggest important features that can be extracted from user reviews. Figure 6, is a sample list of noun phrase that were clustered together based on k-Means results. This information can be beneficial to the domain analyst. Domain analysts will be highlighted regarding important features found for a software, as frequently mentioned by users in the reviews. Development team can take note on these important features for future development of similar software, or for improvements on current software in the next releases.

```
Cluster1:  <<printable coloring activities>>
<<printable activity options>> << printable
coloring pages>>

Cluster2:  <<algebra concepts >> <<algebra
principles>> <<algebraic concepts>>
<<algebra problem>>

Cluster3: <<valuable teaching tool>>
<<traditional classroom>> <<video tutorial>>
```

Figure 7: Clustering Result

This recommendation although not validated for its accuracy (recall and precision), however we believe it can act as an early input to development team or domain analyst to have brief idea pertaining to the main features mentioned by the end users. Additionally, our proposed process can reduce the time for domain analysts to read the entire customer reviews.

Crowdsource requirements engineering in our context depends on the extraction of user reviews; the reviews provided by public that came from various background. The reviews came from their experience using the software as well as the quality of datasets used. This however can be good and bad too. Reviews can be good when users write genuine report relating to the software features such as reporting the frequently used features, reporting features with problems and more. On the other hand, reviews can be useless when users tend to write comments that are not related to software features like sarcasm or harsh words that will not bring added values to the overall reviews. This may have been affected by the use of user reviews extracted from the internet that were not validated. For example, sentences provided by users most of the times did not follow any formatted sentence structure, as what can be expected from Software Requirements Specification documents. The extracted review consists of raw data and free text as it came from large crowd who can freely express their opinion. Therefore, we note this as some of the limitations to the input for crowdsource requirements engineering.

Another thread to validity for our experiment result may be affected by the smaller data set used: only 25 reviews for this small-scale experiment. For that, we agree with [13] on the fact that the accuracy for LSA results will increase if using a larger set of data. Therefore, this will be our immediate future plan to increase the datasets and find a benchmark data for accuracy evaluation.

VI. Conclusion and Future Works

In this paper, the use of software reviews to demonstrate the crowdsource requirements engineering is presented. Firstly, this paper summarized the related works in the area of crowdsource software development and the use of user reviews in requirements engineering research. This is then followed by description on the experiment conducted with FENL tool, supplemented with k-Means clustering by using user reviews as its input, to demonstrate the crowdsource requirements engineering. A promising significant clustering result as indicated by the ANOVA and post hoc Tukey test is obtained and described in the paper. Thus, our contribution from this paper comes in two-fold: demonstrating the crowdsource requirements engineering by using user reviews, and adding data mining techniques with statistical evaluations to Software Engineering research. Although detail experiment being conducted, we believe our experiment may yield a better and a more accurate result in the near future with bigger data sets to work with. Furthermore, we will have to search for a benchmark dataset in this area, so that supplementary accuracy test such as precision and recall can be performed in the near future.

References

[1] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *J. Syst. Softw.*, 2017. vol 126, pp. 57-84.

[2] E. Guzman and W. Maalej, "RE 2014 - Appstore review and reuse," in *How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews*, 2014, pp. 153– 162.

[3] N. Genc-Nayebi and A. Abran, "A systematic literature review: opinion mining studies from mobile app store user reviews," *J. Syst. Softw.*, 2017. vol. 125, pp 207-219.

[4] L. V. G. Carreno, and K. Windbladh. "Analysis of user comments: An approach for software requirements evolution," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 582–591.

[5] J. Howe, "The rise of Crowdsourcing," *Wired*, vol. 14, no. 6, pp. 1-4, 2006.

[6] N. Hasteer, N. Nazir, A. Bansal, and B. K. Murthy, "Crowdsourcing software development: Many benefits many concerns," *Phys. Procedia*, vol. 78, pp. 48–54, 2016.

[7] K.-J. Stol and B. Fitzgerald, "Two's company, three's a crowd: a case study of crowdsourcing software development," in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, pp. 187–198.

[8] T. T. D. Latoza and A. van der Hoek, "Crowdsourcing in software engineering: Models, opportunities, and challenges," *IEEE Softw.*, vol. 33, no. 1, pp. 1–13, 2016.

[9] D. Damian and A. Finkelstein, "StakeSource2.0: Using social networks of stakeholders to identify and prioritise requirements," in *33rd Internation Conference of Software Engineering, ICSE 2011*, 2011, pp. 1022–1024.

[10] S. L. Lim and A. Finkelstein, "StakeRare: Using social networks and collaborative filtering for large-scale requirements elicitation," *IEEE Trans. Softw. Eng.*, vol. 38, no. 3, pp. 707–735, 2012.

[11] A. Adepetu, K. Ahmed, and Y. Al Abd, "CrowdREquire: A Requirements Engineering Crowdsourcing Platform," in *2012 AAAI Spring Symposium Series*, 2012, pp. 1-6, Available at https://www.aaai.org/ocs/index.php/SSS/SSS12/paper/viewFile/4311/4685, Date retrieved: 28/8/2017.

[12] N. H. Bakar, Z. M. Kasirun, N. Salleh, and A. H. A. Halim, "Extracting software features from online reviews to demonstrate requirements reuse in software engineering," in *Proceedings of the International Conference on Computing & Informatics*, 2017, pp. 184-190.

[13] S. Deerwester, S. T. Dumais, G. W. Furnas, and T. K. Landauer, "Indexing by latent semantic analysis," *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, p. 391, 1998.

[14] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. Adaptive C. MIT Press, 2001.