Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology- CCIT 2014.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-20

Towards Requirements Reuse: Identifying Similar Requirements with Latent Semantic Analysis and Clustering Algorithms

Noor Hasrina Bakar, Zarinah M. Kasirun, Hamid A. Jalab

***Abstract*— Software requirements that exist in natural language can easily be understood by various stakeholders. However, when it comes to extracting common requirements from the natural language requirement documents for reuse, manual extraction process can be arduous, expensive, and very error-prone on the results. In this paper, we describe a process of identifying similar requirement documents for reuse in Software Product Lines. Online product reviews were extracted and used as the input mainly due to the scarcity of publicly available requirement documents. Latent Semantic Analysis technique from Information Retrieval was used to identify similar requirement documents and filter out the unrelated ones after the text has been pre-processed. Similar documents were then clustered together by using K-means and Hierarchical Agglomerative Clustering algorithm. As a result, the output from the clustering process will be used to recommend group of related requirement documents to be used in requirement engineering phase for a SPL development.**

Keywords*— *Software Engineering, Requirements Similarities, Requirements Reuse, Software Product Lines, Latent Semantic Analysis, Text Clustering

I. Introduction

Natural language software requirements were proven to be more popular because these forms of documents can easily be comprehended by various stakeholders; either from the technical or nontechnical background. However, when it comes to reusing requirements for a new similar system called the Software Product Lines (SPL), the tasks to manually extract the common and variant features from the natural language requirements can be arduous and error-prone. In addition, a simple survey conducted in [1] revealed that the unavailability of support mechanism and the nonexistence of requirements are among the important factors that hinder software practitioners in Malaysia from practising requirement reuse. This raises the need to have at least a semi-automated extraction process towards publicly available requirements, which can effectively assist the requirements reuse process.

University of Malaya
Malaysia

In the context of SPL, few authors have addressed the solution to this problem. For example, to assist requirements reuse, Niu and Easterbrook [2] were concerned with the verb-direct object to construct functional requirements profile. Yet, this approach is only suitable for organisations that already own accumulated software requirements. Some other forms of natural language requirements that exist (product reviews, online product descriptions, use cases, scenarios, etc.) are not suitable in the case of verb-direct-object extraction. In [3], the ARBOCRAFT tool uses the collection of requirement documents and merges them to create a single system description. Using Hierarchical Agglomerative Clustering, HAC algorithm, the tool produces a hierarchy of features, which are clusters of requirements. Both examples above assume organisations to have complete existing requirements that are easily accessible, in which are not applicable to all organisations. In a more recent research in requirements reuse for SPL, publicly available requirements such as online product reviews[4], brochures[5], or product descriptions[6] are mined for reuse due to the non-existence or unable to reach the actual requirement documents in SPL development.

Applying the HAC, Ferarri et al. used nouns or acronyms to extract common and variant features from publicly available product brochures [5]. They were interested in the features that represented components of the product rather than the functionalities of the product which appeared in [2] & [3]. In a another work that appears in [4], feature descriptors from product summary listed in Softpedia were used as the input to the feature extraction process and Incremental Diffusive Clustering was used in the clustering process.

Inspired by [5] & [4], we will add a contribution to the feature extractions from natural language requirements for reuse in terms of identifying similar requirement documents from online product reviews as a first step towards requirements reuse. Our work employed the technique called Latent Semantic Analysis (LSA) that is followed by clustering algorithms. In Software Engineering area, various authors used the LSA towards various software artefacts as appeared in [7], source code [8], and requirements [9] & [3]. For example, Marcus and Maletic in [8] used LSA to identify the traceability links from system documentation to programme source code. In addition, LSA and Vector Space Model (VSM) were used to identify common features from requirement documents, as appeared in [3], [10], and [9].

Clustering algorithms has been used for grouping similar text [11][12][13]. For the experimental purposes, we use the K-Means and Hierarchical Agglomerative Clustering

Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology- CCIT 2014.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-20

algorithm to group similar requirement documents, in our case textual requirements existed in product review form.

In this paper, we will describe our experiment of hybridizing the LSA and clustering algorithms (LSA+K-Means and LSA+HAC) to identify similar requirement documents. Section II describes the method used in the experiment. In Section III, the result obtained from the experiment will be presented and discussed, and finally Section IV will conclude the paper.

II. Method

Our work is separated into five phases: Fetching Product Descriptions from Internet, Text Pre-processing, Term-Weighting, Identifying Similar Requirement Documents and Clustering Similar Requirement Documents. The following subsections describe each of the phases involved in the experiment.

A. *Fetching Product Descriptions from Internet*

In Phase 1, product descriptions were fetched from the internet. We have used http://www.toptenreviews.com as the site to seek for the product descriptions. For initial experiment purpose, 27 product descriptions were randomly fetched from this website on children learning software, and downloaded into multiple text files.

B. *Text Pre-processing*

In this phase, each document has undergone the pre-processing using Python 2.7 and data cleaning in MSExcel.

TABLE I. TEXT PREPROCESSING

Step	Activity
Step 1	To tokenize sentences into words
Step 2	To remove Stop Words
Step 3	To lemmatize words
Step 4	To count the term occurrences

Processes in TABLE I is modelled in Python 2.7. Sentences are broken down into words via tokenisation process, followed by stop words removal. Stop words are words that occur frequently in a text but do not provide any added value if included. These words include "is", "are", "was", "in", "of", etc., accomplished by applying the following code (Figure 1):

```
raw=raw.lower()
words = re.findall(r'\w+', raw,flags = re.UNICODE |
re.LOCALE)
important_words=[]
for word in words:
   if word not in stopwords.words('english'):
   important_words.append(word)
   important_words = filter(lambda x: x not in
stopwords.words('english'), words)
```

Figure 1. Removing stopwords in Python

Lemmatisation is a process to group together the different inflected forms of a word so they can be analysed as a single item. For example, a word "learning", "learned", and "learns" will be grouped together as "learn". Lemmatisation reduces the word into its canonical form. In this work, we are interested to see nouns because nouns can depict a feature. Features are represented by vector (term frequency t_1 in document n), for example:

V_{Dn}= (tf(t1,Dn),tf(t2,Dn),tf(t3,Dn),…tf(tn,Dn))

Each dimension of document vector is represented by the term in the dictionary. For example, Document 1 (D1) has the text in Figure 2.

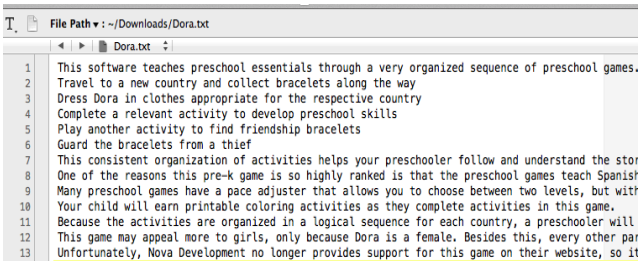


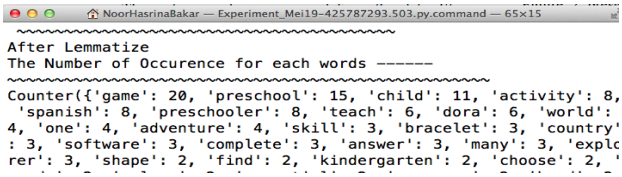
Figure 2. Raw text (product review) fetched from the internet

After pre-processing Step 1 until Step 3, we count the occurrence of the words by using the `Counter()` method from TextBlob[1] library in Python 2.7, as captured in Figure 3.

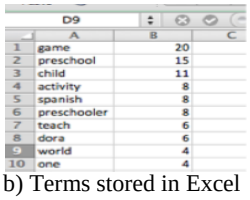
```
for sent in stemmed_words:
processed_words.append(l.lemmatize(sent,'n'))
freqs=Counter(processed_words)
#Counter method to count the occurrence of each
lemmatized words
x=[]
x.append(freqs)
wrtr.writerow(x)  #sending rows to Excel file
```

Figure 3. Using TextBlob in Python to count word occurrences

The data now are in the forms of terms and its count of occurrences (see Figure 4a), but need a cleaning up. The MSExcel functions "Text to Column" and "Paste Special→Transpose" were used to clean up the data for each of the documents (see Figure 4b). Results from preprocessing stage were saved into spreadsheets. Figure 4 presents a snapshop for a sample of extracted terms from D1.



a) Raw output in Python



	A	B	C
1	game	20	
2	preschool	15	
3	child	11	
4	activity	8	
5	spanish	8	
6	preschooler	8	
7	teach	6	
8	dora	6	
9	world	4	
10	one	4	

b) Terms stored in Excel

Figure 4. Extracted terms

**this process is repeated for n number of documents*

[1] http://textblob.readthedocs.org/en/latest/api_reference.html

Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology- CCIT 2014.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-20

C. *Term-weighting (tf-idf)*

The next step (Phase 3) is to calculate the term weighting in each of the documents. For this case, the spreadsheets consisting collection of terms and its occurrences in each document are merged. This produces the term by document matrix, in which can also be seen as vector. Vector Space Model (VSM) is an idea originated in [14] describes algebraic model representing textual information as a vector; the components of this vector could represent the importance of a term (term frequency) or even the absence or presence of it in a document. In VSM, terms that occur in documents are represented as vector of numbers. The original term occurrence will record how many times each term occurs in each of the documents.

The `tf-idf` stands for *term frequency-inverse document frequency*. The `tf-idf` weight is a weight used in information retrieval and text mining, used to evaluate how important a word is to a document in a collection. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the collection [15].

D. *Document Similarity Identification*

In Phase 4, the `tf-idf` values were obtained by a function in MATLAB against the term-document matrix to create a vector space model representation. Vector space representation can be used for information retrieval, classification, clustering, and more. For our case, at this step we are going to perform the LSA to determine document similarity before we can cluster the similar documents.

SVD computation in LSA is what distinguishes the LSA from the more traditional VSM. SVD computation reduces the dimension of the document so that only relevant vectors are considered, while the traditional VSM uses the original dimension of the document which makes it less effective than LSA. Moreover, VSM uses the keyword matching techniques as compared to concept-based techniques that are applied in LSA. The detailed explanation of LSA implementation with the SVD calculation is available in [16]. Singular Value Decomposition (SVD calculation) was applied to reduce the dimension of matrix representation to three different matrices: `S`, `U`, and `V`. In MATLAB R2011a, the following command is used to decompose the term-by-document matrix into matrices `S`, `U`, and `V`:

```
[S U V] = svd(A)
```

Then, matrix `V` was transposed to be V^T. By keeping the dimension to lower dimensions, the SVD computation should bring together the terms with similar co-occurrences. Thus, in our experiment, rank 2 approximation was implemented, so that the first two columns of matrix `U` and `V` are kept:

```
Uk=U(:,1:2)
Vk=V(:,1:2)
```

As a result, rows in V_k contain the coordinates of individual document vectors. These coordinates when projected to an x-y plane will indicate the position of all documents in the problem space (See Figure 5). Document that are far apart indicate the dissimilarities.

E. *Clustering Similar Documents*

In the clustering stage (Phase 5), similar documents are grouped together using two clustering algorithms: the K-Means and the Hierarchical Agglomerative Clustering algorithms.

1) K-Means clustering

The K-means algorithm is a commonly used clustering algorithm with the aim to optimise an objective function (the distance) that is described by the equation:

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

Equation (1)

$\boldsymbol{m_i}$ is the centre of cluster $\boldsymbol{C_i}$, while $\boldsymbol{d(x,m_i)}$ is the Euclidean distance between point $\boldsymbol{x}$ and $\boldsymbol{m_i}$. The following is the algorithm for K-means [17]:

1. Set a fixed number of clusters, c.
2. Randomly pick up a cluster centre.
3. Assign all points in the dataset to the cluster whose centre is the nearest (closest centroid).
4. Recompute the centres for each centroid.
5. Repeat the process in steps 3 & 4 until the centres stop changing.

Although K-Means was pronounced to be one of the simplest clustering algorithms and efficient, there are still problems that exist with the k-means clustering. The obvious one is that the need to know the number of clusters in advance. Additionally, the k-means algorithm tends to go to the local minima that are very sensitive to the starting centroid location. A scatter plot can be produced to view the K-means clustering result.

2) Hierarchical Agglomerative Clustering

To compare the clustering result, we used HAC to group similar product reviews.

HAC was reported to produce better clustering quality [18] [19] as compared to K-means, especially for building document hierarchies. Hierarchical technique produces a series of nested sequence of partitions, starting with a single, all-inclusive cluster at the top which is then split into subclusters at the bottom.

The following is the algorithm for HAC[2]:

1. Identify the similarity between all pairs of documents and present it in a square form (in which ij^{th} entry gives the similarity between the i^{th} and j^{th} document).
2. The most similar document (closer distance value) will be merged to form a cluster.
3. The similarity matrix will be updated to reflect the pairwise similarity between the newly formed cluster and the original documents.
4. Steps 2 and 3 will be repeated until only a single cluster remains.

[2] Complete linkage was used as the cluster distance measure for HAC

Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology- CCIT 2014.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-20

A dendogram is plotted to indicate the result of HAC (See Fig. 7).

III. Results and discussions

We have experimented the method described in Section II towards 27 documents of online product reviews taken from toptenreviews.com. The randomly selected product reviews came from three different product lines in the domain of children educational software: Preschool Learning Software, Algebra Learning Software, and Creative Writing Software. TABLE II shows the number of documents and terms extracted.

TABLE II. DOCUMENTS RETRIEVED FOR THE EXPERIMENT

Software Category	No. of documents	*Total terms
Preschool Learning	10	795
Algebra Learning	7	574
Creative Writing	10	835
Total	27	2204
*selected terms after the pre-processing stage		

a) Similar Documents

All selected documents are projected on the document space, after applying the LSA algorithm towards the *tf-idf* terms weights. As a result, the Vk matric in LSA shows the position of each document in the problem space (the coordinates). The cosine similarity calculation was also applied towards the 27 documents to confirm the distance between each coordinate produced by LSA. From the plot in Figure 6, documents that are far apart imply dissimilarity in which D16 and D25 are the most obvious ones.

a) Cluster of Documents

We then applied the K-Means clustering by initialising 5, 6, and 8 clusters respectively. We then compared the K-Means cluster results with the result produced by the HAC algorithm.

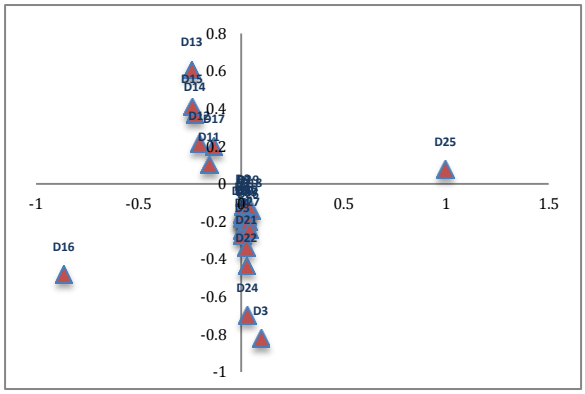


Figure 5. Position of documents in the problem space

The results from both clustering algorithms indicate that D16 and D25 fall apart from other documents as well, while with 8 clusters, k-means discriminate D13, D16, and D25 (TABLE III). Figure 6 and Figure 7 illustrate the clustering results produced by both algorithms.

TABLE III. CLUSTERING RESULTS AFTER APPLYING K-MEANS AND HAC

K-Means		HAC	
C1	D1,D2,D6,D7,D8,D9,D10,D18,D19,D20,D23,D26	C1	D7,D10,D6,D8,D20,D23,D26
C2	D3,D24	C2	D1,D2,D9,D18,D19
C3	D11,D12,D17	C3	D3,D24
C4	D4,D5,D21,D22,D27	C4	D11,D12,D17
C5	D14,D15	C5	D4,D5,D21,D22,D27
C6	***D13***	C6	D13,D14,D15
C7	***D25***	***C7***	***D16***
C8	***D16***	***C8***	***D25***

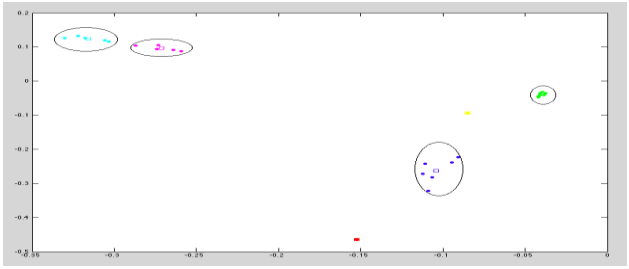


Figure 6. Clusters produced by K-Means algorithm

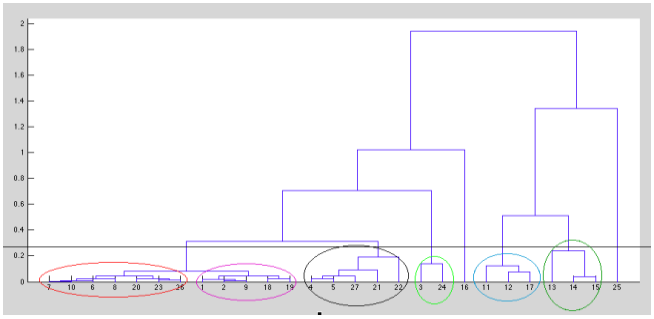


Figure 7: Clusters produced by HAC algorithm

a) Evaluation of Cluster Quality

We used the Average Distance of Documents to cluster Centroid (ADDC) to evaluate the compactness of clusters produced by both algorithms as an internal quality measure. This same measure was also used in [11] and [20] for data clustering problems.

ADDC as the Internal Quality Measure

The following formula is used to obtain the ADDC value, as also used in [11] and [20]:

$$f = \frac{\sum_{i=1}^{N_c} \left\{ \frac{\sum_{j=1}^{p_i} d(o_i, m_{ij})}{p_i} \right\}}{N_c}$$

Equation (2)

where: $\boldsymbol{m_{ij}}$ indicates the $\boldsymbol{j}$th column vector which belongs to cluster $\boldsymbol{i}$,
$\boldsymbol{O_i}$ denotes the centroid vector of the $\boldsymbol{i}$th cluster,
$\boldsymbol{d(O_i, m_i)}$ is the distance between data vector $\boldsymbol{m_{ij}}$ and the cluster centroid $\boldsymbol{O_i}$;
$\boldsymbol{p_i}$ is the number of documents which belongs to the cluster $\boldsymbol{C_i}$; and
$\boldsymbol{N_c}$ stands for the number of clusters

Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology- CCIT 2014.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-20

The lower ADDC value obtained indicates a better clustering quality. As for measuring an external quality, entropy was used to measure the "goodness" for un-nested clusters. The best entropy is obtained when each cluster contains exactly one data point.

TABLE IV. PERFORMANCE COMPARISON WITH REGARD TO CLUSTER COMPACTNESS (ADDC VALUE)

	ADDC value	
	LSA+K-means	LSA+HAC
Cosine	4.0410	3.0686
Euclidean	3.1261	**2.0521**

A good cluster is the one with minimised ADDC value. We used two distance measures for comparison purposes (Cosine and Euclidean). Table IV indicates the ADDC value comparison with regard to the cluster compactness. Clustering solution produced by LSA+HAC produces a lower ADDC value (better cluster), which means that objects within clusters produced by HAC algorithm are more related as compared to the one produced by K-Means algorithm.

Entropy as the External Quality Measure

Entropy is a metric that measures how the various semantic classes are distributed within each cluster. The smallest possible value for entropy is 0.0, which occurs when all symbols in a vector are the same, which means a perfect clustering solution. We adopted Shannon's Entropy calculation as follows:

$$H(X) = \sum_{i=1}^{n} P(x_i) * \log_2(P(x_i)) \quad \textit{Equation (3)}$$

where:

n = number of clusters

P(xi)=probability that a member belongs to class ***xi***

For the entropy value, our experiment indicates that LSA+K-means produce slightly better entropy value (2.403) as compared to LSA+HAC (2.7408). However, when comparing our entropy result with the one reported in [9], our entropy values were higher than theirs. The study in [9] used requirement or specification documents which were validated, but in our case we used the online product review that is raw and un-validated. This could explain the huge difference between entropy values obtained.

From the experiment conducted, we observe that both clustering algorithms discriminated almost similar documents that are not related (refer Table 3). Although product descriptions taken from the toptenreviews.com are already domain specific, our experiment indicated that there are still documents that are dissimilar.

As for evaluating the clustering quality, the LSA+HAC produces a better ADDC value for the internal cluster quality, taking the consideration to use the Cosine and Euclidean distance measure in the ADDC formula. For both distance measures, LSA+HAC produces better internal cluster quality. This is similar with the result obtained by Cui et al. in [11] in their experiment. Authors in [9] used HAC + LSA to compare the performance with HAC + VSM. Their purpose is to compare the effect of adopting LSA and VSM that is combined with HAC clustering. The result from their experiment indicated that the use of HAC + VSM produced a lower entropy value, in which it outperformed the algorithm that uses HAC + LSA. This is because LSA will only obtain a better performance in a larger corpus. In their case the experiment only used 28 and 59 requirements[3] – a smaller corpus compared to ours.

From this observation, at this moment it is not easy to tell which clustering algorithm is better. Obtaining good clustering results is always dependent on the input parameters (the data sets) [21]. For example, to get a good clustering result with k-means, an optimum number of clusters to be created is needed to be taken into consideration: with the question to ask is what is the optimal number of clusters to use when applying the k-means. To get a better view of the external cluster validation, we might need to add the F-measure, NMI-Measure, and Purity, thus compare the results obtained rather than relying on the Entropy alone.

IV. Conclusion and future works

In this paper, we have described an approach to identify similar requirements that appear in natural language extracted from online product reviews. We have used the product reviews obtained from the toptenreviews.com as an input to our experiment. We have obtained the initial result of implementing LSA to find similar documents, which were then seeded into K-means and HAC algorithm. The clusters produced are evaluated using average distance as an internal compactness measure and by using the Shannon's entropy formula for evaluating the external quality measure.

In the near future, we are going to seed a larger number of documents provided by the LSA into an optimisation algorithm, the Particle Swarm Optimisation (PSO), since this optimisation algorithm will not require us to specify initial number of clusters to use. Requirement documents that are clustered together will act as recommendation for requirements engineer prior to selecting features for a particular software product line. Although product descriptions taken from the internet are already domain specific, our experiment shows that there are still documents that are dissimilar. We believe having the initial filtering stage by using LSA combined with clustering can be a significant contribution for identifying common requirements from natural language requirements for reuse in the Software Product Lines area.

Acknowledgment

This research project is supported by Ministry of Higher Education Malaysia, Grant # FP050 / 2013A.

References

[1] N. H. ; Bakar and Z. M. Kasirun, "Exploring Software Practitioners Perceptions and Experience in Requirements Reuse An Survey in Malaysia," *Int. J. Softw. Eng. Technol.*, vol. 1, no. 2, 2014.

[3] the number of requirements (28 and 59) in this sentence means the functionality, which only contains one or two sentences. In our case, we use 27 documents (product reviews), where in each of the documents, usually more than five sentences are used.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-20

[2] N. Niu and S. Easterbrook, "Extracting and Modeling Product Line Functional Requirements," *2008 16th IEEE Int. Requir. Eng. Conf.*, pp. 155–164, Sep. 2008.

[3] N. Weston, R. Chitchyan, and A. Rashid, "A Framework for Constructing Semantically Composable Feature Models from Natural Language Requirements," in *Software Product Lines Conference*, 2009, pp. 211–220.

[4] H. Hariri, C. Castro-Herera, M. Mirarkholi, J. Cleland-Huang, and B. Mobasher, "Supporting Domain Analysis Through Mining and Recommending features from ONline Product Listings," *IEEE Trans. Softw. Eng.*, vol. 39, no. 12, pp. 1736–1752, 2013.

[5] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta, "Mining commonalities and variabilities from natural language documents," in *Proceedings of the 17th International Software Product Line Conference on - SPLC '13*, 2013, p. 116.

[6] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," *Proc. 2013 9th Jt. Meet. Found. Softw. Eng. - ESEC/FSE 2013*, p. 290, 2013.

[7] H. Kruger and P. S. Kritzinger, "Software Traceability using Latent Semantic Analysis and Relevance Feedback."

[8] a. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," *25th Int. Conf. Softw. Eng. 2003. Proceedings.*, pp. 125–135, 2003.

[9] 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," *2008 12th Int. Softw. Prod. Line Conf.*, pp. 67–76, Sep. 2008.

[10] K. Kumaki, R. Tsuchiya, H. Washizaki, and Y. Fukazawa, "Supporting commonality and variability analysis of requirements and structural models," *Proc. 16th Int. Softw. Prod. Line Conf. - SPLC '12 -volume 1*, p. 115, 2012.

[11] X. Cui, T. E. Potok, and P. Palathingal, "Document clustering using particle swarm optimization," in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, 2005, pp. 185 – 191.

[12] E. Hasanzadeh, M. Poyan, and H. A. Rokny, "Text clustering on latent semantic indexing with particle swarm optimization (PSO) algorithm," *Int. J. Phys. Sci.*, vol. 7, no. 1, pp. 116–120, Jan. 2012.

[13] K. Chen, W. Zhang, H. Zhao, and H. Mei, "An approach to constructing feature models based on requirements clustering," *13th IEEE Int. Conf. Requir. Eng.*, pp. 31–40, 2005.

[14] G. Salton, A. Wong, and C. S. Yang, "Vector Space Model for Automatic Indexing," *Commun. ACM*, vol. 18, no. 11, 1975.

[15] G. Salton and C. Buckley, "Term Weighting Approaches in Automatic Text Retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513 – 523, 1988.

[16] 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.

[17] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*, Adaptive C. MIT Press, 2001, p. 461.

[18] R. C. Dubes and A. K. Jain, *Algorithms for Clustering Data*. Englewood Cliffs, N.J.: Prentice Hall, 1988.

[19] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," in *Proc. KDD-2000 Workshop TextMining*, 2000.

[20] van der D. Merwe and A. Engelbrecht, "Data clustering using particle swarm optimization," in *Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003)*, 2003, pp. 215 – 220.

[21] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz, "Internal versus External cluster validation indexes," *Int. J. Comput. Commun.*, vol. 5, no. 1, 2011.

About Author (s):

Noor Hasrina Bakar received her BSc (Information Technology) from Marquette University in Milwaukee, Wisconsin in 1998, and MSc (Computer Science) from University of Malaya in 2009. Currently, she is a full-time PhD student in Software Engineering Department at the Faculty of Computer Science and Information Technology, University of Malaya. Her current research is in the area of feature extractions from requirements for reuse in software product lines.

Zarinah Kasirun is an Associate Professor in Software Engineering Department at the Faculty of Computer Science and Information Technology, University of Malaya. Her research interests include requirements engineering, requirements visualization, software metrics and quality, and software product line engineering.

Hamid Jalab received his B.S. degree in electrical engineering from the University of Technology, Baghdad, Iraq, and his M.S. and Ph.D. degrees in computer system from the Odessa National Polytechnic University, Odessa, Ukraine. He currently works as a Senior Lecturer in the Faculty of Computer Science and Information Technology at the University of Malaya, Kuala Lumpur, Malaysia. His research interests include mathematical computing, signal processing, digital image processing, wavelets, neural networks, and image retrieval.