Research Article

International Journal of Advanced Computer Research, Vol 8(39)
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
http://dx.doi.org/10.19101/IJACR.2018.839013

Automatic clustering of bug reports

Maen Hammad[1*], Ruba Alzyoudi[2] and Ahmed Fawzi Otoom[1]
Associate Professor, Department of Software Engineering, The Hashemite University, Zarqa, Jordan[1]
Lab's Supervisor, Information Communication & E-Learning Technology Center, The Hashemite University, Jordan[2]

Received: 18-September-2018; Revised: 06-November-2018; Accepted: 08-November-2018

Abstract
It is widely accepted that most development cost is spent for maintenance and most of the maintenance cost is spent on comprehension. Maintainers need to understand the current status of the code before updating it. For this reason, they examine pervious change requests and previous code changes to understand how the current code was evolved. The problem that faces them is how to locate related previous change requests that handled a specific feature or topic in the code. Quickly locating previous related change requests help developers to quickly understand the current status of the code and hence reduce the maintenance cost which is our ultimate goal. This paper proposes an automated technique to identify related previous change requests stored in bug reports. The technique is based on clustering bug reports based on their textual similarities. The result of the clustering is disjoint clusters of related bug reports that have common issues, topic or feature. A set of terms is extracted from each cluster, as tags, to help maintainers to understand the issue, topic or feature handled by the bug reports in the cluster. An experimental study is applied and discussed, followed by manual evaluation of the bug reports in the generated clusters.

Keywords
Software maintenance, Bug reports, Clustering, Textual similarities.

1.Introduction
One known fact about software systems that they have evolved as a result of continuous maintenance activities. They are subject to change in order to fix a bug, enhance functionality, or adapt to new environmental changes. It is widely accepted that large portion of the cost of a software system over its lifetime is spent on maintenance activities [1] that also include comprehension. Understanding the current status of the code after many maintenance activities is challenging. Therefore, maintainers need to go back and examine all code changes and the reasons behind each code change. One way to do this is by examining all change requests stored in bug reports for a specific issue or component.

Problems
For bug tracking systems, it is not easy to identify all bug reports that are related to a specific issue in the software. For example, it is hard to answer the following question. What are all bug reports that handled the login feature in the software? In this paper, we focus on reducing the cost of corrective maintenance by helping developers to understand and identify related bug reports.

Developers and users of the system report bugs. Bug reports need to be analyzed in order to determine the type of the bug, its location, its applicability, and the developer who can fix it. Manual inspection and exploring a large number of bug reports to solve the above critical issues consumes time and effort of developers. As a result, the corrective maintenance cost will increase. The key to handle the above issues is to identify the similarity links between different bug reports. Based on the identified similarity, bug reports can be grouped and organized into related groups. In this case, maintainers can quickly determine similar reports and identify the type of similarity for each group.

Benefits
Identifying related bug reports can support corrective maintenance activities in:

- Helping to identify similar bug reports that may need to be merged.
- Helping in the developer assignment process. All related bug may be assigned to the same developer(s).

*Author for correspondence

- Helping to understand all bug reports that affected specific feature or component.
- Helping to keep the bug reports categorized in the repository.

About the proposed approach

The proposed approach utilizes the agglomerative hierarchical clustering (AHC) to generate a set of disjoint clusters of bug reports. The approach also tags each cluster with representative keywords that reflect the features to topics handled by the cluster. The clustering is performed based on the textual similarities between bug reports. Each bug report is preprocessed by tokenizing, stop words removal and stemming. Then, each bug report is represented as a vector of tokens. Jaccard coefficient is used for the similarity calculations between vectors. Finally, most frequent terms appeared in each cluster are extracted as representative tags for the cluster.

The significance of this work is the full automation of the approach without applying any supervised techniques. Textual information is automatically extracted and preprocessed from bug reports, automatically clustered, and tags are automatically extracted. The AHC has been utilized in the clustering because it fits well with this type of problems where each item is a set of textual keywords.

Related work

The related work can be divided into three categories; bug report classification, bug reports clustering and applying AHC on software.

The work in the area of bug report classification focuses on classifying bug reports into specific categories. Most of these approaches are machine learning approaches that use predefined features. Antoniol et al. [2] found that it is possible to classify bug reports into corrective maintenance and other kinds of activities based on the text on the issues posted in bug tracking systems. Thung et al. [3] categorized bug reports into three categories; control and data flow, structural, and non-functional. Pandey et al. [4] applied machine learning algorithms to classify issue reports into bug and non-bug categories. Another technique is presented by Pingclasai et al. [5]. They utilized topic modelling and apply three classification techniques. Menzies and Marcus [6] classified bug reports based on their severity. They built a tool named severity issue assessment (SEVERIS) to assign severity levels to defect reports. Their approach is based on text mining and machine learning methods. Podgurski et al. [7] presented a failure reports classification technique that uses supervised and unsupervised pattern classification and multivariate visualization. These techniques are applied to profiles of failed executions. The goal is to group together failures with the same or similar causes. Zanetti et al. [8] proposed a bug reports calcification method based on measures to quantify the social embedding of bug reporters in the collaborative networks. Other machine learning classification techniques are presented in [9−12]. Nagwani and Verma [13] proposed CLUBAS (Classification of Software Bugs Using Bug Attribute Similarity). It combines text clustering, frequent term calculations and taxonomic term mapping techniques. The algorithm CLUBAS is an example of classification using clustering techniques. In Nagwani et al. [14], a methodology is presented to find the taxonomic terms using latent Dirichlet allocation (LDA) for software bug classification.

Researchers in the field of Bug reports clustering worked on identifying related bug reports based on their contents. Our work differs in applying the AHC on bug reports with Jaccard similarity coefficient. Limsettho et al. [15, 16] automatically grouped bug reports based on their textual similarity. Their approach also labels groups with meaningful names. For clustering, they used expectation maximization (EM) and x-means algorithm. Topic modelling was applied to project bug reports into topic vector space. Sandusky et al. [17] studied the related bug reported found by constructing a bug report network (BRN) that is created by community members. Fry and Weimer [18] proposed automatic defect reports clustering to identify similar bugs. The goal is to reduce maintenance cost by triaging and fixing similar bugs in aggregate. Their approach utilizes syntactic and structural information available in static bug reports. They used metrics for information retrieval and natural language processing fields. They also defined their own metrics to compare individual defect report sub-components. Rus et al. [19] clustered defect reports based on their summary and description fields. For clustering, they applied k-means, normalized cut, and size regularized cut. Kumaresh and Baskaran [20] proposed an ontology-assisted Semi-supervised clustering based classification for bug reports. Gopalan and Krishna [21] proposed clustering based method to identify duplicate bug reports. They adapted a clustering algorithm named incremental clustering (INCLUS) algorithm. Most researchers who applied AHC in

software engineering were for the purpose of architecture recovery, decomposition and reengineering. Cui and Chae [22] analyzed agglomerative hierarchical clustering algorithms in software reengineering activities. Rousidis and Tjortjis [23] presented a methodology using AHC for information extracted from Java source code. The aim is to capture the program structure and to support comprehension. Hammad [24] applied agglomerative hierarchical clustering to identify related commits in software repositories. The goal is to recover traceability links between related code changes. Maqbool and Bari [25, 26] studied hierarchical clustering techniques in the context of software architecture recovery and modularization. They also presented an automatic cluster labelling scheme using identifiers. Keywords within identifiers are ranked using frequency and inverse frequency ranking schemes.

The work differs from the related work in the area by utilizing the AHC algorithm to cluster bug reports into disjoint clusters and by tagging each cluster by a set of keywords extracted from each cluster. The approach is unsupervised, automated and easy to apply.

This paper is organized as follows. Section 2 presents the proposed approach with all its techniques. A case study on two projects is discussed in Section 3 with the evaluation process and results. Dissections of the results with the threats that may affect the validity are presented in Section 5. Conclusions and our future work are presented in Section 6.

2.Materials and Methods

Our goal is to support corrective maintenance activities by proposing an approach to identify and categorize similar bug reports via clustering. We specifically apply the agglomerative hierarchical clustering on bug reports. Similarity is measured based on the textual content of bug reports.

The textual contents of bug reports are processed before they are used in the similarity calculations and clustering. The main processing steps are stemming and stop word removal. Bug reports will be extracted from large open source projects. The results will be disjoint groups (clusters) of bug reports.

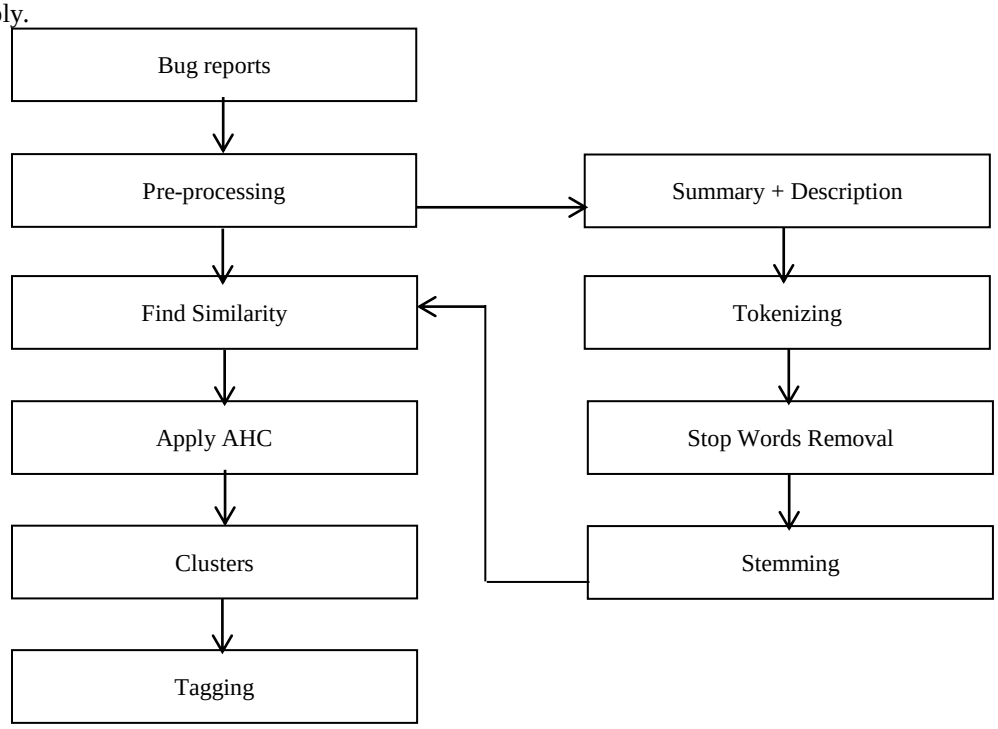

Figure 1 The main steps of the proposed approach

2.1Overview of the approach

The steps of the proposed approach are shown in *Figure 1*. The process starts with a set of bug reports extracted from a bug tracking system or repository. In the next step, bug reports are pre-processed as follows:

1. Extract the textual summary and description of each bug report.

2. Remove stop words from the extracted summary and description
3. Find the stem for the remaining tokens.
4. Represent each bug report as a vector of tokens.

The similarity among all vectors is calculated using the Jaccard similarity coefficient as a necessary step for the AHC algorithm. The result of clustering will be disjoint clusters of bug reports.

2.2Pre-processing

In this step, bug reports are transformed into vectors of keywords. Each vector represents a bug report. Two textual contents of the bug report are extracted; summary and description. The text is tokenized in to keywords (or tokens). In the next step, stop words are removed. All words that do not contribute in differentiating a collection of documents are removed (i.e., “t”, “that”,” the” “they”, “these”, and so on).

For example, *Figure 2* shows the summary and description for three bug reports from the bug reports of the open source project AspectJ. Each bug report is represented as a set of keywords. The final pre-processing step is stemming. In stemming, various forms of a word are converted to the base word. *Figure 3* shows the three bug reports in *Figure 2* after applying all pre-processing steps.

Bug 420210 Support additional message insert keys in declare error/warning It would be good to be able to insert the enclosing class name or enclosing member for a joinpoint.

Bug 419279 ajc option to change -Xlint level per-message without Xlintfile The -Xlintfile option is not a great fit for controlling message across multiple build projects, specifically in my case from the pluginManagement section of a maven parent pom. The problem is that you need a local file to configure the per-message output levels (ignore/warning/error) when you really want to specify it in the build script or in a shared file. As an alternative to -Xlintfile, it would be handy to be able to change an Xlint warning level per message using command line options. For example: ajc -Xlint:adviceDidNotMatch=ignore would override the XlintDefault.properties file for the adviceDidNotMatch message. With Regards Rob

Bug 418129 Can't introduce annotation onto introduced method from trait-patterned aspect

Figure 2 Summary and description for three bug reports from AspectJ

420210;Support;addit;messag;insert;key;declar;error;warn;good;abl;insert;enclos;class;name;enclos;membe r;joinpoint;

419279;option;chang;Xlint;level;per;messag;Xlintfil;Xlintfil;option;gre;fit;control;messag;across;multipl;bu ild;project;specif;case;pluginManag;section;maven;parent;pom;problem;need;local;file;configur;per;messag ;output;level;ignor;warn;error;realli;want;specifi;build;script;share;file;altern;Xlintfil;hi;abl;chang;Xlint;war n;level;per;messag;use;comm;line;option;exampl;Xlint;adviceDidNotMch;ignor;overrid;XlintDefault;prope rti;file;adviceDidNotMch;messag;Regard;Rob;

418129;introduc;annot;introduc;method;trait;ptern;aspect;

Figure 3 Result after applying all the pre-processing steps on the three bug reports shown in Figure 2

2.3Clustering

Our approach utilizes clustering techniques to identify related bugs. The goal of clustering is to group related bugs with similar summary and description into disjoint similar clusters. So, we decided to apply the AHC. The main advantage of the AHC algorithm is its outcome. The outcome is disjoint clusters of bug reports. This can be an essential step in identifying groups of bug reports that handle the specific issue in the system or specific feature. So, the archived bug reports are categorized and grouped into sets of meaningful and related groups. Once each bug report is represented as a vector of keywords, clustering can be easily applied. AHC is a bottom up clustering technique. It starts by finding the most similar pairs and group them into clusters. Next, the similarity between clusters from the previous step is computed and new larger clusters are generated. The process continues in a hierarchical way until it ends up with one cluster that contains all elements. The generated hierarchical structure is called dendrogram. So, a cut is done at a similarity threshold value in the dendrogram.

A similarity measure is needed to measure how two vectors of keywords (bug reports) are close to each

other. This is necessary for clustering. The Jaccard similarity coefficient measure has been used to calculate similarities between vectors of bug reports. For two vectors of keywords V1 and V2, the similarity is measured by the number of common elements between the two vectors (intersections) divided by the number of unique elements in both vectors (union). Similarity = $|V1 \cap V2| / |V1 \cup V2|$.
For example, consider two sets A = {0,1,2,3,8,9} and B = {0,1,3,4,5,6,8,9}. How similar are A and B? The Jaccard similarity coefficient is 5/9 (0.55). A two dimensional similarity matrix is generated for all vectors. The matrix stores the similarity values between all vectors. The similarity measure for each vector with all other vectors.

2.4Term frequencies

After getting clusters of bug reports, our next goal is to get a meaningful summary or tags that reflect the topics or features covered by each cluster. Our approach utilizes the most frequent tokens appeared in each cluster. We believe that repeatedly using specific terms in a cluster is an indication for the features or topics handled by this cluster. Processed vectors (tokenized and stemmed) are used to find the occurrences of each term in each vector. As a result, a term frequency matrix is generated for all vectors. This matrix contains all terms that appeared in all vectors and their occurrences. It is used to extract most frequent terms for each cluster.

2.5Tags extractions

Our next goal is to tag each generated cluster with a set of keywords. These keywords or tags help maintainers to:

- Categorize clusters into meaningful categories.
- Identify topics, features or issues handled by each cluster of bug reports.
- Categorize new arriving bug reports into one of the existing categories.

We propose to select the tags from the top frequent terms in bug reports of the cluster. For each cluster, all term frequencies are identified. This includes all terms that have appeared at least once in any bug report in the cluster. All terms with their frequencies are listed. Then, top N frequent terms are reported as tags for the cluster.

Actually, N is a threshold value that can be determined based on different factors; size of the cluster, number of terms, and occurrences. Our goal is to use a small N value that represents large coverage of terms. Section 3 details the experimental results we conducted. Based on manual inspections of some generated clusters, we determined N to be three. So, the most three frequent terms are selected as tags for the cluster. In case more than one term has the same number of occurrences, all of them are selected.

3.Results

In this section, we report the results of applying the proposed approach on real datasets and the evaluation of these results.

3.1Dataset

The dataset under consideration is a collection of bug reports for the two open source projects AspectJ and Tomcat. The dataset of these projects can be obtained online and has four more projects other than AspectJ and Tomcat. The URL is: (http://figshare.com/articles/The_dataset_of_six_open_source_Java_projects/951967). We mainly extracted summary and description of each bug report for the two projects. *Table 1* lists the extracted information after the pre-processing step. The first column lists the projects' names. The second column is the number of extracted bug reports for each project within the time period shown in the third column. The total number of tokens (without stop-words) is shown in the fourth column.

The number of clusters resulted from applying the AHC on the 593 bug reports from AspectJ is 60. The number of clusters resulted from clustering the 1056 bug reports of Tomcat project is 93. *Figures 4 and 5* show the dendrograms in the clustering results for both projects. The dendrogram show how the AHC algorithm works, to group or cluster the bug reports level by level. The AHC has successfully clustered all the bug reports.

Table 2 shows the size distribution of each generated cluster. Sizes of clusters vary. Some clusters have many bug reports while others have only one report. For example, the maximum cluster size for AspectJ has 148 bug reports followed by cluster with 147 bug reports. On the other hand, there are 10 clusters that have only one bug report.

Table 1 Extracted data from AspectJ and Tomcat

Project	#Bug reports	Time period	# Tokens
AspectJ	593	2003 -2013	69606
Tomcat	1056	2002-2014	53667

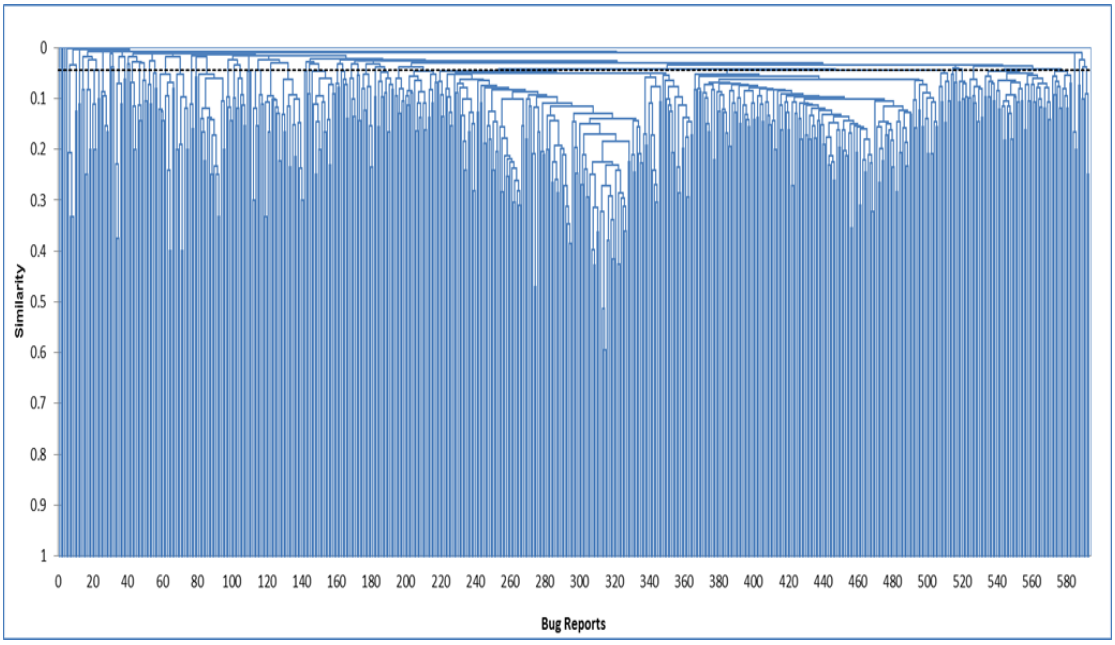

Figure 4 The generated dendrogram from clustering AspectJ reports

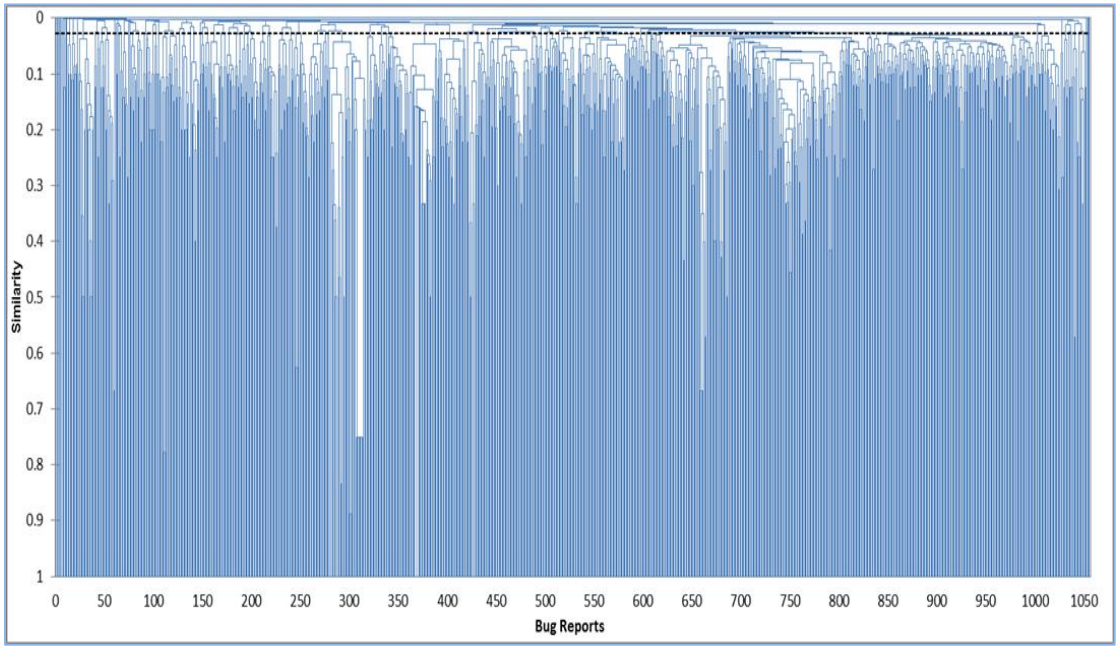

Figure 5 The generated dendrogram from clustering Tomact reports

Table 2 Sizes of clusters for both projects

Size	#Clusters AspectJ	#Clusters Tomcat	Size	#Clusters AspectJ	Clusters Tomcat
1	10	18	16	1	1
2	13	9	18	0	2
3	10	9	19	1	1
4	5	3	21	0	1
5	4	8	24	0	1
6	3	11	29	0	2
7	2	3	38	0	1
8	1	4	39	1	1
9	1	4	42	0	1
10	3	6	43	0	1
11	0	2	147	1	0
12	1	0	148	1	0
13	1	1	172	0	1
14	1	1	205	0	1

For example, one cluster has four bug reports. The summary and description of these bug reports are shown in *Table 3*. It is clear that three of the bug reports talk about the same issue which is "*ajdoc*".

Cluster with only bug report means they have unique or very few terms. Wrong spelling and using abbreviations also affect clustering. *Table 4* shows seven examples of clusters with one bug report in both projects. For each bug, the description is missing. Moreover, developers used long variable names that were handled as long tokens. Abbreviations and numbers were also used in the summary that affects the similarity calculations.

Table 3 Summary and description of bug reports for one cluster

Summary + Description
Bug 391123 Added support for more cache backing(s
Bug 82755 [ajdoc] update ajdoc to support Java 5 language features; Java 5 langauge features such as enums and annotations need to be supported by ajdoc.
Bug 71811 AJDoc: should be able to set encoding like javadoc ajdoc doesn't support the -encoding -docencoding and -charset options of javadoc. Especially, it doesn't pass on the -encoding to ajc. If your project uses a source file encoding (e.g. UTF-8) differing form the platform standard, and some java identifiers use characters byond the standard ASCII range, you won't be able to use ajdoc at all.
Bug 68494 ajdoc does not support .aj files In addition to accepting .java source files ajdoc needs to accept .aj files.

Table 4 Sample of seven clusters with only one bug report

Project	Bug_ID	Summary
AspectJ	389456	NPE in EclipseTypeMunger.mungeNewMethod
	152873	Optimize shouldWeaveAnnotationStyleAspect with Patch
	145322	Failure of testCompareSubclassDelegates() on J9 1.5.0 SR1
Tomcat	55184	NPE in PojoMethodMapping getMessageHandler
	43435	bstractReplicatedMap.memberDisappeared is executed more than the necessity.
	54241	NPE in BodyContentImpl
	49655	ExpressionFactoryImpl.createMethodExpression is not EL 2.2 Spec conform

For each cluster, a set of keywords or tags are extracted to reflect the topics or features handled by the cluster. We propose to extract top N most frequent terms. These most frequent terms are considered as tags. In our approach, we select the most top three (N=3) terms as tags for the cluster. In case there are one or more terms with equal occurrences, all of them are included. So, the number of tags may exceed three. For example, the tags for the cluster in *Figure 6* are; "file", "source", "need", "java", "ajdoc" and "support". These terms have top three occurrences.

We need to investigate the percentage of tags to terms in all clusters. This is because we need to know the coverage of tags in terms based on our threshold value (N=3). This threshold was set to three, based on analysing the results of the processed data for the two projects. We analysed different N values and concluded on value three. The smaller number of representative and meaningful tags is our target. For each cluster, the number of tags was divided by the total number of terms (# of tags / # of terms) to get the percentage coverage of tags over terms. Then, the average of all percentages was calculated. The results are:

- The average percentage coverage of tags to terms in AspectJ is 39%.
- The average percentage coverage of tags to terms in Tomcat is 44%.

Based on these results, we can conclude that our threshold, in average, covers at least 39% of terms. This result supports our goal in selecting a small number of tags (only three) with large percentage coverage.

3.2 Evaluation

Our aim is to evaluate if the tags reflect meaningful references to the topics handled by the bug reports of the clusters. For these reasons, we decided to prepare an evaluation process that includes human experts for both the clusters and the tags of the clusters. The number of participants in the evaluation was 40 human experts. Participants hold a bachelor or master degree in computer science and work in different IT fields. We randomly selected 15% of clusters which resulted in 10 clusters from AspectJ and 15 clusters from Tomcat with 25 clusters in total. Both description and summary of all bug reports in the same cluster were shown to participants. They had to evaluate if the bug reports from their point of view are Related, Weakly Related and Not Related. Tags were also shown in the cluster with another question about how much they reflect the topics of the cluster (Reflect, Weakly Reflect and Not Reflect). All the 25 clusters were shown together to participants. The first 10 clusters were for AspectJ.

As shown in the chart in *Figure 6* for AspectJ and Tomcat clusters, related responses exceed both not related and weakly related responses. The evaluation concluded that they are not related to clusters 2, 8, 14, 18 and 20. When we went back to them, we found these clusters have many of bug reports more than others and each bug report has large textual contents. This may be the reason. Participants might lose focus for large bug reports. Another justification would be that AHC may not provide good results with bug reports with large text. Further investigation needed to clarify this issue. Clusters 5, 9, 10, 15 and 19 were weakly related. We found these clusters have many bug reports more than others, but with short text. The same observation is noticed in *Figure 7* for the same clusters. It shows responses to the tag reflection question. On clusters 2, 8, 14, 18 and 20, the tags for these clusters do not reflect the cluster topic based on responses. In conclusion, most participants found that generated clusters have related bug reports and their tags reflect the contents of each cluster.

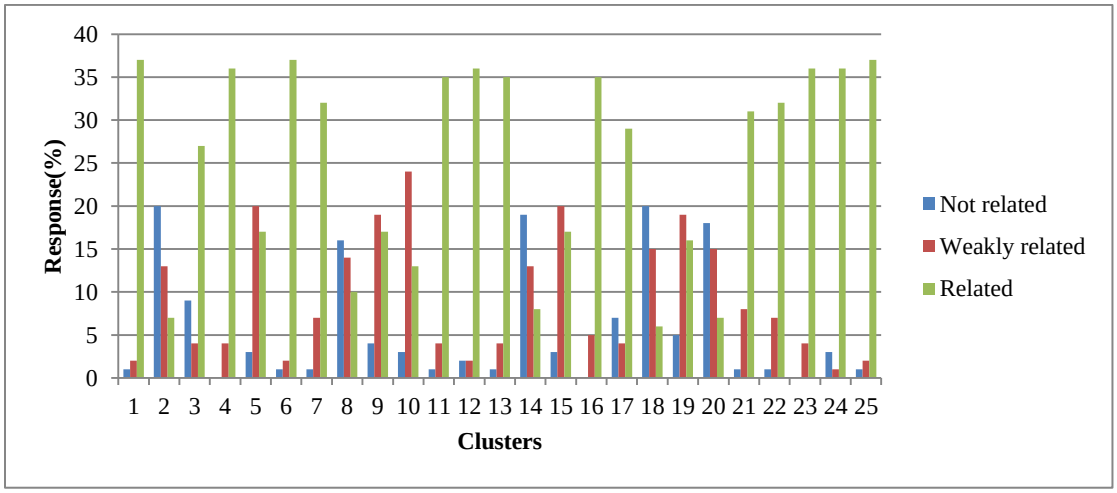

Figure 6 Responses for related bugs in the clusters of AspectJ and Tomcat

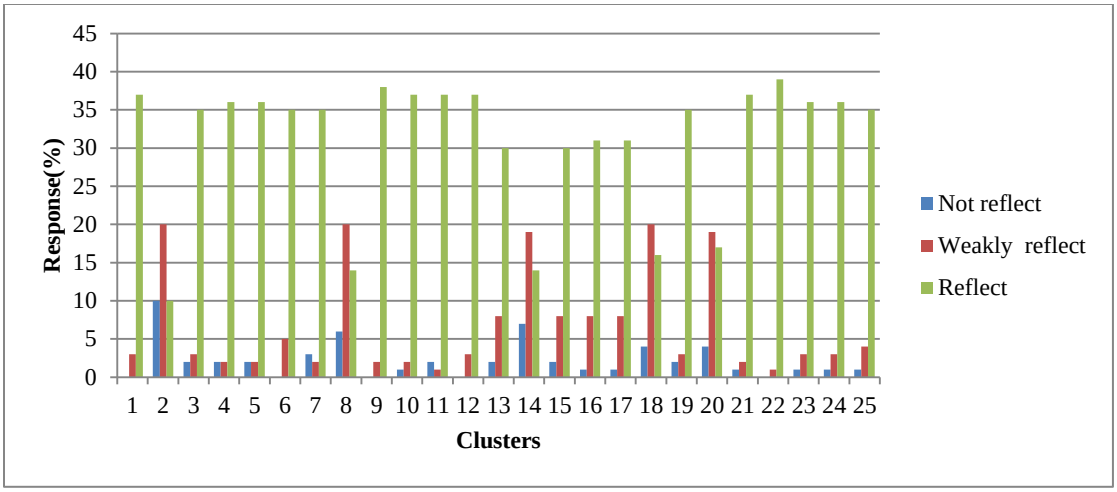

Figure 7 Responses for the reflection of tags for AspectJ and Tomcat

4.Discussion

Based on the exponential and evaluation results presented in section 3, we can highlight three observations. First, the proposed approach succeeded in cluster bug reports into disjoint sets of clusters. So, bug reports can be clustered based on their textual contents that are written in the summary and the description of bug reports. The second observation resulted from the generated clusters. Each cluster has many common keywords that can be utilized to generate a label, a tag or a summary for each cluster. Finally, the generated clusters and tags are related based on human experts. This observation means that clusters can be used in future maintenance activities in helping developers as listed above. The clustering and evaluation results of this paper are mainly based on two open source projects. The results may change if other or larger datasets were used. The similarity cut point was set to be dynamic. The number of resulted clusters is affected by this threshold value. Another factor may affect the results is the number of top frequent terms that are selected as tags for clusters. The number was set to three. Results are subject to change in case this number is changed or the selection criteria are changed. Finally, the manual evaluation by experts was done on subset and randomly selected clusters.

5.Conclusion and future work

Identifying related bug reports reduces maintenance cost by helping maintainers locate and understand related change request. Moreover, categorization of clusters is also of a great importance of reflecting the features covered by each cluster. In this paper, an automated approach has been proposed to identify related bug reports. The approach is based on clustering archived bug reports based on their textual similarities. For each cluster, a set of keywords is extracted to reflect the topic or issues handled by the bug reports in the same cluster. The AHC has been utilized with the Jaccard similarity coefficient to perform clustering based on applying the proposed technique on bug reports of six projects. Results showed that bug reports can be clustered based on their textual similarities. Moreover, based on human verification, most clusters have related bug reports and their tags reflect the contents of each cluster. The future work aims to extend the approach to handle new reported bug reports. The goal is automatically cluster the newly reported bug into one of the already existing clusters. Apply topic mining techniques on clusters to extract more useful information about each cluster is another possibility extends under consideration. We are currently working on different datasets and planning to do more experiments with different threshold values for the number of tags.

Conflicts of interest

The authors have no conflicts of interest to declare.

References

[1] Schach SR. Object-oriented and classical software engineering. New York: McGraw-Hill; 2007.

[2] Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc YG. Is it a bug or an enhancement?: a text-based approach to classify change requests. In proceedings of the conference of the center for advanced studies on collaborative research: meeting of minds 2008. ACM.

[3] Thung F, Lo D, Jiang L. Automatic defect categorization. In working conference on reverse engineering 2012 (pp. 205-14). IEEE.

[4] Pandey N, Sanyal DK, Hudait A, Sen A. Automated classification of software issue reports using machine

learning techniques: an empirical study. Innovations in Systems and Software Engineering. 2017; 13(4):279-97.
[5] Pingclasai N, Hata H, Matsumoto KI. Classifying bug reports to bugs and other requests using topic modeling. In Asia-pacific software engineering conference 2013 (pp. 13-8). IEEE.
[6] Menzies T, Marcus A. Automated severity assessment of software defect reports. In international conference on software maintenance 2008 (pp. 346-55). IEEE.
[7] Podgurski A, Leon D, Francis P, Masri W, Minch M, Sun J, et al. Automated support for classifying software failure reports. In proceedings of international conference on software engineering 2003 (pp. 465-75). IEEE.
[8] Zanetti MS, Scholtes I, Tessone CJ, Schweitzer F. Categorizing bugs with social networks: a case study on four open source software communities. In proceedings of the international conference on software engineering 2013 (pp. 1032-41). IEEE Press.
[9] Kanwal J, Maqbool O. Bug prioritization to facilitate bug report triage. Journal of Computer Science and Technology. 2012; 27(2):397-412.
[10] Lamkanfi A, Demeyer S, Soetens QD, Verdonck T. Comparing mining algorithms for predicting the severity of a reported bug. In European conference on software maintenance and reengineering 2011 (pp. 249-58). IEEE.
[11] Otoom AF, Al-Shdaifat D, Hammad M, Abdallah EE. Severity prediction of software bugs. In international conference on information and communication systems 2016 (pp. 92-5). IEEE.
[12] Zhou Y, Tong Y, Gu R, Gall H. Combining text mining and data mining for bug report classification. Journal of Software: Evolution and Process. 2016; 28(3):150-76.
[13] Nagwani NK, Verma S. CLUBAS: an algorithm and java based tool for software bug classification using bug attributes similarities. Journal of Software Engineering and Applications. 2012; 5(6):436-47.
[14] Nagwani NK, Verma S, Mehta KK. Generating taxonomic terms for software bug classification by utilizing topic models based on latent Dirichlet allocation. In international conference on ICT and knowledge engineering 2013(pp. 1-5). IEEE.
[15] Limsettho N, Hata H, Monden A, Matsumoto K. Automatic unsupervised bug report categorization. In international workshop on empirical software engineering in practice 2014 (pp. 7-12). IEEE.
[16] Limsettho N, Hata H, Monden A, Matsumoto K. Unsupervised bug report categorization using clustering and labeling algorithm. International Journal of Software Engineering and Knowledge Engineering. 2016; 26(7):1027-53.
[17] Sandusky RJ, Gasser L, Ripoche G. Bug report networks: varieties, strategies, and impacts in F/OSS development community. International conference on software engineering workshop 2004 (pp. 80-4).
[18] Fry ZP, Weimer W. Clustering static analysis defect reports to reduce maintenance costs. In working conference on reverse engineering 2013 (pp. 282-91). IEEE.
[19] Rus V, Nan X, Shiva SG, Chen Y. Clustering of defect reports using graph partitioning algorithms. In SEKE 2009 (pp. 442-5).
[20] Kumaresh S, Baskaran R. Ontology assisted semi-supervised bug report classification. Journal of Theoretical & Applied Information Technology. 2015; 78(2):170-80.
[21] Gopalan RP, Krishna A. Duplicate bug report detection using clustering. In Australian software engineering conference 2014 (pp. 104-9). IEEE.
[22] Cui JF, Chae HS. Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. Information and Software Technology. 2011; 53(6):601-14.
[23] Rousidis D, Tjortjis C. Clustering data retrieved from java source code to support software maintenance: a case study. In European conference on software maintenance and reengineering 2005 (pp. 276-9). IEEE.
[24] Hammad M. Identifying related commits from software repositories. International Journal of Computer Applications in Technology. 2015; 51(3):212-8.
[25] Maqbool O, Babri H. Hierarchical clustering for software architecture recovery. IEEE Transactions on Software Engineering. 2007; 33(11):759-80.
[26] Maqbool O, Babri HA. Automated software clustering: an insight using cluster labels. Journal of Systems and Software. 2006; 79(11):1632-48.

Maen Hammad is an Associate Professor in the Software Engineering Department at The Hashemite University, Jordan. He completed his Ph.D. in Computer Science at Kent State University, USA in 2010. He received his Master in computer science from Al-Yarmouk University-Jordan and his B.S. in Computer Science from The Hashemite University-Jordan. His research interest is Software Engineering with focus on Software Evolution and Maintenance, Program Comprehension and Mining Software Repositories.
Email: mhammad@hu.edu.jo

Ruba Khalil Al-Zyoudi is a Computer & Virtual E-Learning Lab's Supervisor in the Information Communication & E-Learning Technology Center at the Hashemite University. She Received Her Master degree In Software Engineering from The Hashemite University, Jordan, in 2018. She received her B.S in Software Engineering, from the Hashemite University-Jordan in 2006. Her research interest is Software Engineering with focus on Software Evolution and Maintenance, Program Comprehension, Mining Software Repositories, Machine Learning and Software Development.
Email: ruba.alzyoudi@hu.edu.jo

Ahmed Fawzi Otoom is currently working as an Associate professor in the Software Engineering department at The Hashemite University, Jordan. He has a PhD degree in Computer Science from the University of Technology, Sydney (UTS), Australia, 2010. He received his BS in Computer Science from Jordan University of Science and Technology, Jordan, and an MS in Software Engineering from the University Western Sydney, Australia, in 2002 and 2003, respectively. His main research interests include Pattern Recognition Techniques and its applications in the areas of Software Engineering and Image Analysis.
Email: aotoom@hu.edu.jo