**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**

***IJCSMC, Vol. 5, Issue. 2, February 2016, pg.268 – 273***

# SUMMARIZATION OF BUG REPORTS USING FEATURE EXTRACTION

**R.Nithya[1], A.Arunkumar[2]**

[1]PG Scholar, Department of Computer Science and Engineering, Sri Krishna College of Engineering and Technology, Coimbatore, India

[2]Assistant Professor, Department of Computer Science and Engineering, Sri Krishna College of Engineering and Technology, Coimbatore, India

[1] nithyarangasamy03@gmail.com; [2] arunkumara@skcet.ac.in

## Abstract

**Recent years in IT industry, the practical need for automatic summarization has increased to a large extend. This paper attempts to fill this void by providing a comprehensive overview of research in summarization, by including the more traditional efforts in sentence extraction. Automatic text summarization is based on linguistic and empirical methods which analyzes how frequently certain key words were present in the bug repository. Automatic Text Summarization is a technique where a process or application summarizes a text. A text is given to the application and the application returns a less redundant extract of the original text. So far automatic text summarization has not yet reached the quality that was possible with manual summarization. In order to reduce developer's time and efforts when reviewing a bug report, the proposed prototype also provides an extractive summary visualization of each bug report. Sentence features are considered and weighted to evaluate each sentence and based on the feature values sentences are selected to form the summary. The proposed approach increases the quality of the summary and also detects the duplicate bug reports in the document.**

**Keywords: Software Engineering, Bug Report Analysis, Summarization of Software Artifacts, Bug Report Duplicate Detection.**

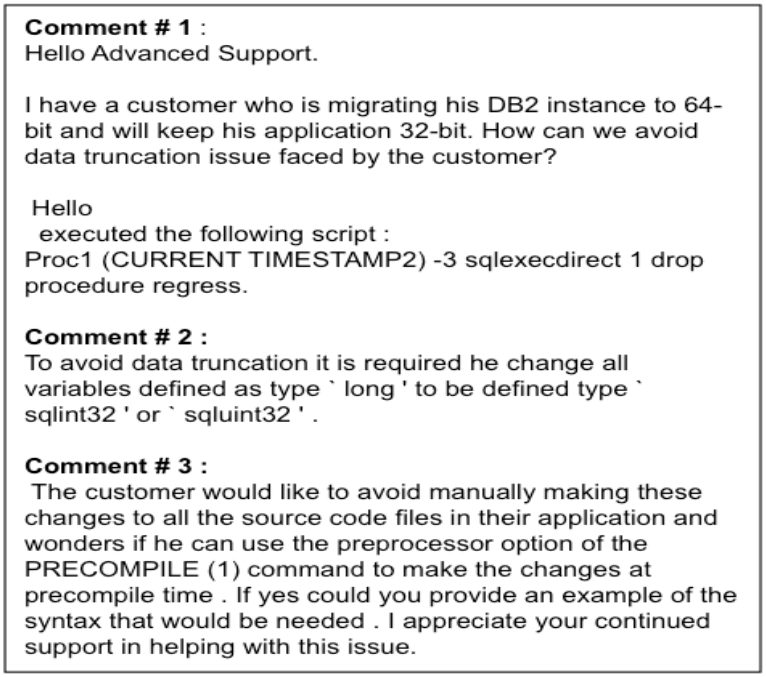
## I. INTRODUCTION

As new software systems are getting more complex and greater every day, software bugs are unavoidable phenomenon. Bug repositories aid software developers to communicate about a software development project among themselves, and in open-source projects, with corresponding users of the software. The bugs reported by developers or users may include defects linked with deployed software as well as desired enhancements to the

system. Bug reports vary in length. Some are short, consisting of only a few words. Others are lengthy and include conversations between many developers and user. For larger software projects, it is common that bug reports are categorized to allow efficient handling by the right subset of the software development team. Considerable amount of communication might take place among the reporter of the bug and the developers through the bug report. As a result, bug reports often appear like recorded conversation messages from multiple people structured in sequence where each message is composed of various paragraphs of unstructured text. Automatic summarization of bug reports is a technique to condense the quantity of data a developer might need to go through. Many developers put considerable amount of effort for finding and debugging software bugs. Automatic bug report summarization has two approaches: extractive summarization and abstractive summarization[5,6]. An extractive summarization method selects subset of sentences from the input document to create a summary. An abstractive summarization method builds an internal semantic representation of the text and then applies natural language processing techniques to create a summary. It has been observed that a bug report submitted by a tester is often a duplicate (two bug reports are supposed to be duplicates if they express the same issue or problem and thereby have the same solution to fix the problem) of an existing bug report. Studies illustrate that the percentage of duplicate bug reports can be up-to 25-30%. This can considerably slow down the bug fixing process and product release [1]. Figure 1 is a sample bug report and its corresponding summary.

Figure 1: Example of Bug Reports and its Summary



**Comment # 1** :
Hello Advanced Support.

I have a customer who is migrating his DB2 instance to 64-bit and will keep his application 32-bit. How can we avoid data truncation issue faced by the customer?

Hello
executed the following script :
Proc1 (CURRENT TIMESTAMP2) -3 sqlexecdirect 1 drop procedure regress.

**Comment # 2 :**
To avoid data truncation it is required he change all variables defined as type ` long ' to be defined type ` sqlint32 ' or ` sqluint32 ' .

**Comment # 3 :**
The customer would like to avoid manually making these changes to all the source code files in their application and wonders if he can use the preprocessor option of the PRECOMPILE (1) command to make the changes at precompile time . If yes could you provide an example of the syntax that would be needed . I appreciate your continued support in helping with this issue.

**Sample Bug Report**

I have a customer who is migrating his DB2 instance to 64-bit and will keep his application 32-bit.

To avoid data truncation it is required he change all variables defined as type ` long ' to be defined type ` sqlint32 ' or ` sqluint32 ' .

The customer would like to avoid manually making these changes to all the source code files in their application and wonders if he can use the preprocessor option of the PRECOMPILE (1) command to make the changes at precompile time .

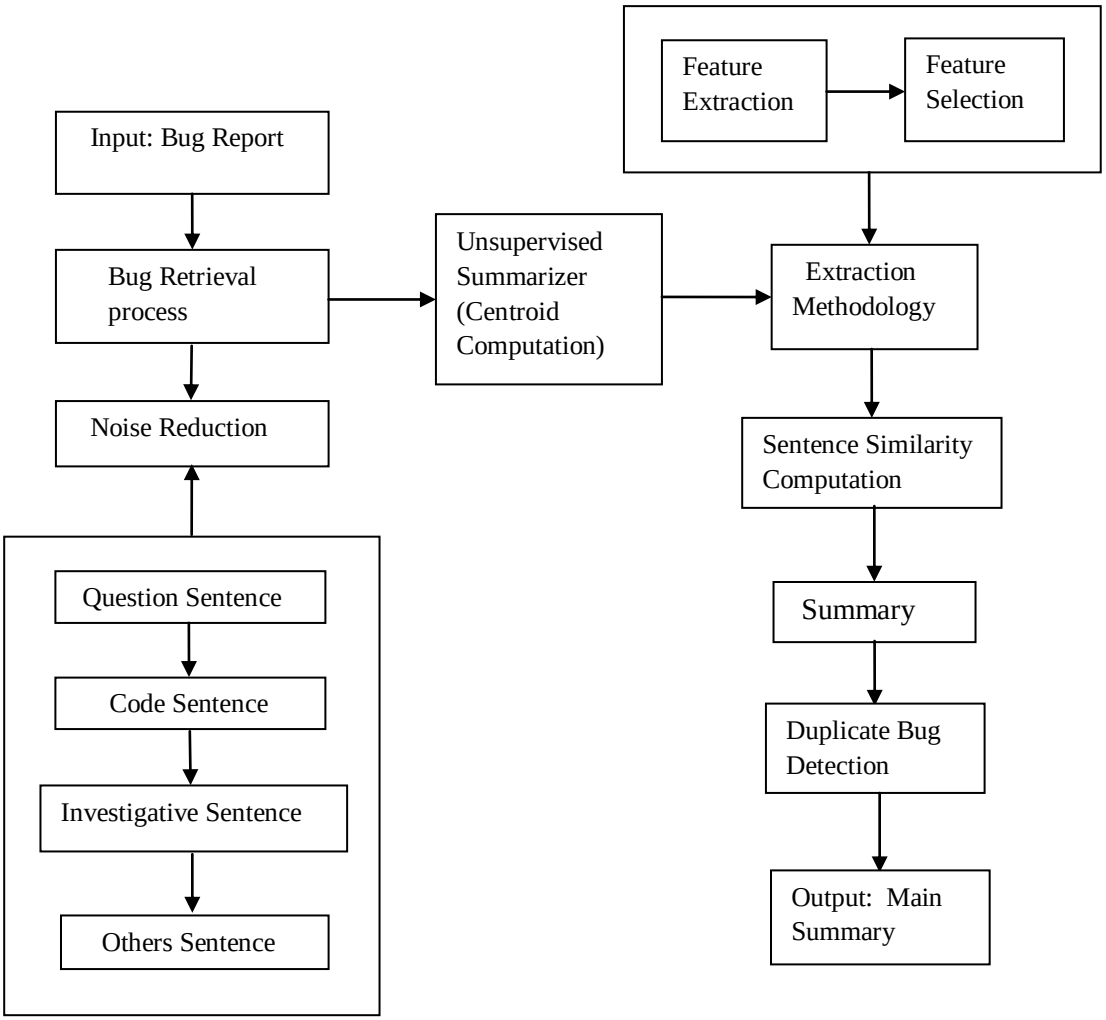
**Bug Report Summary (30% of sentences)**

## II. PROPOSED SYSTEM

For a bug that needs to be summarized, initially the bug reports are passed to the noise reduction module. Each sentence in the bug report is classified into question, investigative sentence, code fragment and other sentences. In this approach, different type of sentences are filtered out and pass the filtered set to a summarizer, which applies the unsupervised techniques to extract the summary from the set of valuable sentences. The filtered content is accepted to the summarizer module. In the next module, Extractive summarization method selects subset of sentences from the input document to create a summary. It maintains the redundancy by extracting the appropriate features from the document[2]. Features are the characteristics of the sentence like position, length. The sentences having high feature values are selected to form the summary. Then the similarity between the sentences in the bug report measured. At last duplicate bug reports in the repository are removed by comparing summaries of the bug reports. Figure 2 represents the architecture of the proposed report summarization system.

Figure 2: Summarization Of Bug Reports Using Feature Extraction – Architecture Diagram



**A. Noise Reduction**

In order to filter out the unwanted information in the bug reports, each bug report is passed into the noise reducer. In the noise reducer the sentences in the bug reports are classified into four categories[4].

- Question sentences: the problem being reported will be described in these sentences. These sentences usually contains words like what, why, how.
- Code sentences: these sentences contain code fragments, stack traces or command outputs that the user might have provided as part of initial bug description, or as part of investigation or solution.
- Investigative sentences: these sentences convey some options that the user can try to further investigate the issue or give more details on understanding the problem. These sentences would typically hold application specific information often indicated by occurrence of domain specific keywords.
- Other sentences: These sentences are very often greeting sentences, for example "Hello", "thank you for your support".

**B. Centroid**

Unsupervised summarization refers to the way of summarizing a text or document without using a trained model. This has the evident advantage of not requiring any labeled data, which is usually difficult and/or costly to collect because of high manual effort involved[7]. An extractive summarization refers to any summarization method in which the sentences in the summary are chosen from the input document and are used as it is given in the input. Unsupervised summarization methods work by selecting sentences that are central to the input document. This centrality can be measured by using the 'centroid' method. Then, the sentences chosen for the summary are based on their distance from the Centroid value.

Centroid method is used to measure the weight of each sentence and categorize the sentence with is more specific to the bug report. In this, each unit of text is represented as a weighted vector of TF and IDF (Term Frequency times Inverse Document Frequency). The unit of text is a sentence from the bug report in the repository. The method then proceeds by finding a centroid sentence, which is a pseudo-sentence whose vector has a weight equal to the average weight of all the sentence vectors in the bug report. Sentences that contain words from the centroid are more pinpointing of the topic of the document and hence are chosen in the summary. For each sentence Si, the method defines a term called Centroid Value of Si, which is measured by adding the corresponding weights in the centroid sentence, of the terms in Si. Once the centroid values of sentences have been calculated, the summary is constructed by choosing sentences in the declining order of their centroid values.

**C. Extraction methodology**

Extractive summaries are formed by extracting sentences with high feature values. In extraction methodology, first keywords in the bug report are extracted by using term frequency and inverse document frequency metric for each term in the input document[3]. The feature values of the sentences in the bug report indicate the probability of the evaluated sentence in the summary[8]. To form the bug report summary, the sentences are sorted based on the probability values in the descending order of the feature values. Features that are measured in the extraction methodology are categorizes into four major groups: structural, participant, length, lexical features.

- Structural features are related to the conversational structure of the bug reports in the repository. Example, the position of the chosen sentence in the comment and the position of the sentence in the bug report.
- Participant features are associated to the conversation participants. For example if the sentence is made by the same person who filed the bug report.
- Length features indicates the length of the sentence normalized by the length of the longest sentence in the comment and also normalized by the length of the longest sentence in the bug report.
- Lexical features are related to the amount of unique words/keywords in the sentence.

Once sentences with high values are selected, similarities between these sentences are measured. Sentences with minimum similarity with other sentences in the bug report are selected and included in the summary.

**Algorithm to Extract Keywords**

**Input:** Set $B$ – set of abstracts in text format; total number of abstracts $O = |B|$ $N$ – size of the training set of abstracts
*Separator-List* = { , . ; - ' < > ( ) [ ] / " : }

**Auxiliary Variables:** Training-Set $T$ – set of $M$ abstracts randomly selected from $B$; $T \subseteq B$
*Accept-List* – set of keywords acceptable as secondary keywords *Non-Accept-List* – set of words not acceptable as secondary keywords *User-Choice* = {0 or 1} for a word *Already-Accounted-Words* – set of keywords in *Accept-List* already accounted for their presence in a particular abstract

**Output:** Set *Freq-Sec-K* – set storing the frequency of appearance of each word in the *Accept-List* in the decreasing order of the frequency of their appearance in the $N$ abstracts

**Initialization:** *Accept-List*
*Non-Accept-List*
Training-Set $T$

Begin *Self-Extract-Key-Words*
// Form the training set of abstracts
1. while ($|T| < N$) do
2. Generate a random integer $n \in [1 \ldots N]$
3. if $B[n] \notin T$ then
4. $T \quad T \cup B[n]$
5. end if
6. end while
// Build the *Accept-List* and *Non-Accept-List*
7. for every abstract $Q \in T$ do
8. for every word $X \in Q$ do
9. if $X \notin$ *Separator-List* then
10. if $X \notin$ *Accept-List* AND $X \notin$ *Non-Accept-List* then
11. *User-choice* Ask user to enter 0 to put the word in *Non-Accept-List* or 1 for *Accept-List*
12. if *User-choice* = 1 then
13. *Accept-List* *Accept-List* V $\{X\}$
14. else
15. *Non-Accept-List* *Non-Accept-List* V $\{X\}$
16. end if
17. end if
18. end if
19. end for
20. end for
// Count the frequency of appearance of the words in *Accept-List*
21. for every word $X \in$ *Accept-List*
22. *Freq-Sec-K*($X$) 0
23. end for
24. for every abstract $R \in B$ do
25. *Already-Accounted-Words*
26. for every word $X \in R$ do
27. if $X \notin$ *Separator-List* AND $X \notin$ *Non-Accept-List* then
28. if $X \notin$ *Already-Accounted-Words* AND $X \in$ *Accept-List* then
29. *Freq-Sec-K*($X$) *Freq-Sec-K*($X$) + 1
30. *Already-Accounted-Words* *Already-Accounted-Words* V $\{X\}$
31. end if
32. end if
33. end for
34. end for
35. Sort the words in *Freq-Sec-K* in the decreasing order of the frequency of appearance of the secondary keywords
return *Freq-Sec-K*
End *Self-Extract-Key-Words*

**D. Duplicate bug detection**

In bug repository, one bug may be reported and stored more than once. These duplicate bugs are removed by comparing with other bug reports in the repository. Each report is compared to the ones submitted previous to it. Least Common Subsequence score is computed for each report pair[9]. Taking the highest score as the indication of identical bug reports. In this way, bug reports with least score are removed from the repository.

## III. CONCLUSION AND FUTURE WORK

In this paper we have created an automatic, unsupervised, bug report summarization approach that should be applicable to different bug tracking systems. It enable users to find more relevant documents more accurately, with less need to consult the full text of the document Future work is to see if we can improve the precision of summarization approach so as to be able to auto-extract Frequently Asked Questions from a bug repository.

## REFERENCES

[1] C. Sun, D. Lo, S.C. Khoo and J.Jiang, "Towards More Accurate Retrieval of Duplicate Bug Reports," Proc. 26th Int'1 Conf. Automated Software Eng. (ASE'11), pp.253-262, 2011.

[2] S. Rastkar, G.C. Murphy and G. Murray, "Summarizing Software Artifacts: A Case Study of Bug Reports, " Proc. 32nd Int'1 Conf. Software Eng. (ICSE '10), vol. 1, pp. 505-514, 2010.

[3] Nenkova and K. McKeown, "Automatic Summarization," Foundations and Trends in Information Retrieval, vol. 5, no. 2/3, pp. 103-233, 2011.

[4] S. Mani, R. Catherine, V.S. Sinha and A. Dubey, "AUSUM: Approach for Unsupervised Bug Report Summarization." Proc. ACM SIGSOFT 20th Int'1 Symp. the Foundations of Software Eng. (FSE '12), article 11, 2011.

[5] Pimpalshende. A. N, "Overview of Text Summarization Extractive Techniques," IJECS Volume 2, Issue 4, pp. 1205-1214, 2013.

[6] Neelima Bhatia, ArunimaJaiswal, "Trends in Extractive and Abstractive Techniques inText Summarization." International Journal of Computer Applications (0975-8887) Volume 117- No.6, May 2015.

[7] Wael H. Gomaa, Aly A. Fahmy, "A Survey of Text Similarity Approaches," International Journal of Computer Applications (0975-8887) Volume 68- No.13, April 2013.

[8] MasoumehZareapoor, Seeja K. R, "Feature Extraction or Feature Selection for Text Classification: A Case Study on Phishing Email Detection," I.J Information Engineering and Electronic Business, 2015, 60-65 Published Online march 2015 in MECS.

[9] Sean Banerjee, BojanCukic, Donald Adjeroh, "Automated Duplicate Bug Report Classification using Subsequence Matching," IEEE 14th International symposium on High –Assurance Systems Engineering, pp. 74-81, 2012.