Vol.2, No.6 ISSN Number (online): 2454-9614

An Objective Based Approach to Bug Report Summarization

K.D. Chandran [1], Prof. A. Ananda Rao [2], P. Radhika Raju [3]
[1] *M.Tech Scholar in Department of CSE, JNTUA college of Engineering, Ananthapuramu, India.*
kdchandran.cse@gmail.com
[2] *Professor in Department of CSE, JNTUA college of Engineering, Ananthapuramu, India.*
akepogu@gmail.com
[3] *Lecturer in Department of CSE, JNTUA college of Engineering, Ananthapuramu, India.*
radhikaraju.p@gmail.com

***Abstract*—So many software artifacts were created and maintained as part of the software development process. As software developers working on a project they interact with several artifacts, one such artifact is a bug report. Bug reports generally maintained in a bug repository. The bug reports are used to help the developers to understand how previous bugs are resolved and understand multiple reasons for specific bug. In order to understand the bug report by developers, they need to go through large amounts of text, this process may consume more time. This problem can be solved by automatically creating the summaries of existing bug reports so that developers go through the summaries instead of the entire report. This short summary saves the developers time and help to understand and resolve bugs quickly. The present work produces the summary of extracts based on the objective of sentences using an objective based approach instead of existing classifier based approaches. This approach produces the summary better and minimizes the noise when compared to the existing approaches and reduce the requirement of huge corpus.**

Keywords—Summarization of bug reports, objective based approach, Summarizatio of software artifacts.

I. Introduction

Today's world, many organizations will maintain bug repositories to store the bunch of bug reports, which is useful for developers in the future. A software developer working on a project they often visit the bug reports to understand the root causes of specific bug and how previous actions have been taken to resolve the problem. D. Cubranic et al. [1] Proposed a search recommendation system which can help a developer to identify similar bugs from a bug repository. The developer needs to go through all recommended bug reports to identify the useful information relevant to what the developer wants. Trawling through a flood of data for all recommended bugs might consume more time. The developers still need to follow the monotonous process. Murray et al. [2] suggested that when a bug report is resolved and closed, its respective author should write the summary of abstracts manually. This abstracted summary is helpful for developers who will visit the bug report in the future and allow them to a better understanding of the bug. However in this method, the developer who creates the abstracted summary wants to read the all conversations which are taking place between several stakeholders. So, it takes a lot of time to go through the flood of text and need human resources more.

Because of the dynamic nature of a bug report and the requirement of human resources, it is not an optimal solution and not working in practice. Therefore, there is a need for automatic summarization.

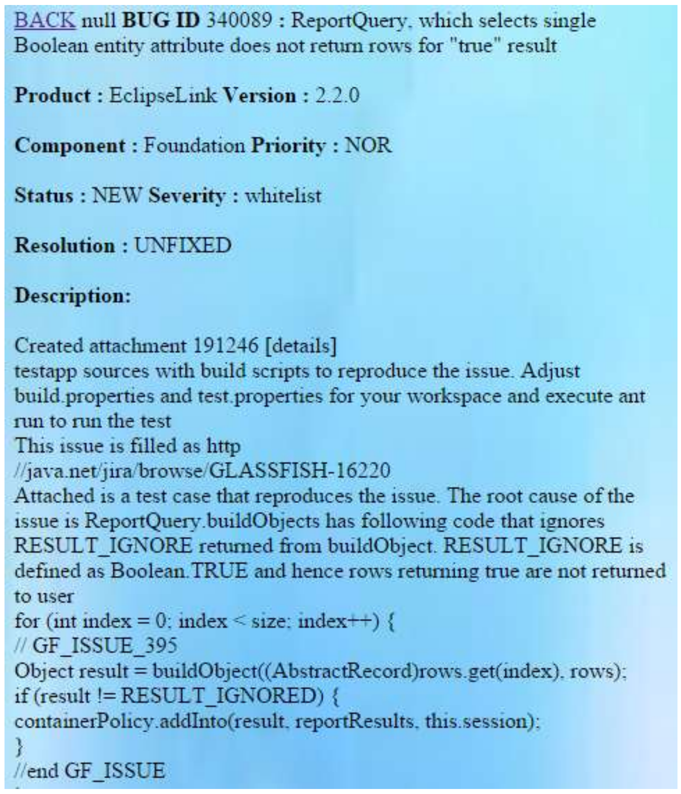
Sarah Rastakar et al. [3] proposed a supervised learning approach to bug report summarization. Using this approach they evaluate different summarizers which are trained on the bug report corpus and email corpus to produce summaries for bug reports as well as for email threads. These summarizers use a classifier based approach to produce the summary. A corpus was needed to train the classifier and the corpus was prepared by selecting important sentences by participants or annotators manually. This process needs more human resources and it is a time consuming process. To overcome the above problem, it is necessary to develop a summarizer which reduces the requirement of huge corpus.

Herein we produce the summary using objective based approach. The bug reports summarized based on the objective, such as whether a sentence indicates a question, answer, suggestion, code and other type of sentences. In addition to the objective of sentences and another criteria called positional feature was also considered to generate the summary. Here, positional feature is useful whenever more sentences having the same weight, based on the position summarizer decides which sentence should come first and which is next. Generally, a bug report is a conversation between several stakeholders in terms of questions and answers. To create the summary, we just concentrate on these conversations instead of all fields in the bug report. The bug report template example is as shown in Fig 1. This approach was applied on 15 bug reports related to Java technology from a Bugzilla[1] bug management system. On these bug reports, our approach marks precision 0.59 and recall 0.38. This result shows that our summarizing approach produces a better summary than existing approaches. The main contributions of the work are as follows,

- Avoiding preparation of the huge corpus by classifying each sentence automatically using an objective based approach.
- Produce good summary, which is easily understandable and useful to developers.
- Experimental evaluation of the summarization process and compares the results of an objective based approach with existing approaches.
- Finally, a survey was performed to know our generated bug summaries take how much time to read and how it is understandable over the original bug reports.

The summary is generated based on weight and the positional feature of a sentence. A user study was conducted where participants are requested to rate the summary in terms of how much time taken to read and understand. The results show that our summary performs better when compared to the original bug report.

1. http://bugs.eclipse.org



BACK null **BUG ID** 340089 : ReportQuery, which selects single Boolean entity attribute does not return rows for "true" result

Product : EclipseLink **Version :** 2.2.0

Component : Foundation **Priority :** NOR

Status : NEW **Severity :** whitelist

Resolution : UNFIXED

Description:

Created attachment 191246 [details]
testapp sources with build scripts to reproduce the issue. Adjust build.properties and test.properties for your workspace and execute ant run to run the test
This issue is filled as http //java.net/jira/browse/GLASSFISH-16220
Attached is a test case that reproduces the issue. The root cause of the issue is ReportQuery.buildObjects has following code that ignores RESULT_IGNORE returned from buildObject. RESULT_IGNORE is defined as Boolean.TRUE and hence rows returning true are not returned to user

```
for (int index = 0; index < size; index++) {
// GF_ISSUE_395
Object result = buildObject((AbstractRecord)rows.get(index), rows);
if (result != RESULT_IGNORED) {
containerPolicy.addInto(result, reportResults, this.session);
}
//end GF_ISSUE
```

Fig.1. An example of the sample bug report structure.

II. Related Reasearch

According to E. Hovy et al. [4] a summary is a significant portion of one or more texts and that is no longer than half of the original texts. The above definition states that a summary should not exceed 50% of the length of the original texts. Murray et al. [2] States that system generated summary must close to the human generated abstract summary. Automated text summarization aims to provide a condensed version of representation of information that the user wants to get.

Summarization techniques can be roughly grouped into three categories based on data processing methods. They are statistical, general linguistic and knowledge based approaches. The statistical approach summarizes without understanding about data, and relies certain statistical features. Recent work includes the classification method produces a summary by including sentences based on the training data. The knowledge based approach, process the text using domain knowledge as well as Natural Language Processing (NLP) techniques. The general linguistic approach works in a general domain, but rely on natural language techniques.

Based on the output, summarization techniques are classified into two types: extraction and abstraction. Using extraction technique, the summary can be produced by simply extracting the important sentences from the original text(s) based on the statistical and linguistic features of a text. Extractive methods work by selecting a subset of phrases or sentences that exist in the original text to form the summary. The extraction approach performed in three steps: (1) Ask the annotators to summarize the set of documents which is used as a corpus to train the system, (2) Develop a model to summarize the document with some statistical features, (3) Produce summary based on the model and feature of the contents in the document. Single document summarization can be mostly performed using an extraction method. Abstraction technique produces the summary based on the pattern for a respective sentence using NLP techniques. Abstraction technique involves semantic based text summarization. In this paper, we produce the extractive summary by identifying the objective of sentences.

Summarizing a single document is called single document summarization, while produce a single summary from a set of documents is called multi document summarization. Multi document summarization is difficult when compared to a single document summarization. In this paper single document summarization is performed.

Rapid use of electronic data, it is necessary to create the summary. Many research works has been taken to produce the summary to speech and texts. X. Zhu et al. [5] develop a system to produce the summary to telephone conversations using speech acts. Many summarizing approaches have been discussed in the literature [6]. The supervised learning approach produces summaries based on training data on both input and output data using different classifiers. The unsupervised learning approach produces the summary based on input data and statistical properties. Gabriel Murray et al. [7] developed a system to produce an abstractive summary of meeting conversation based on ontology mapping. In a formative study, they found that automatic summaries produced by this system are significantly better than human selected extracts on usability and coherence criteria. Alok Ranjan et al. [8] developed an approach to automatic summarization of text using simplified Lesk algorithm and WordNet without considering the conversational features like the position of the word or sentence and input etc.

Summarizing of general text(s) is different from the domain specific document like a bug report. Because the format of a bug report varies depending on the system used to store the bug report. For example, the format of bug report in the Bugzilla bug management system is different from the KDE bug repository. Except the time stamped comments between several stakeholders remaining are fixed fields with default or pre-defined values such as a status field indicates bug is in which state, bug name resembles what kind of bug is?, and what it indicates. Optional fixed fields are varied from one system to another, except the necessary fixed fields. In some bug reporting systems comments once written are allowed to edit but some other systems are not. So, the only time stamped comments are considered to produce the summary and leave the fixed fields.

Bug repositories maintain a flood of reports, efforts have been taken by many researchers to improve the content of the report and making bug report management easier. In some bug management systems comments once submitted are not allowed to edit, because of this reason duplicate bug reports are generated. Wang et al. [9] Proposed an approach to detect duplicate bug reports using both NLP and execution traces. Some bug management systems simply allow editing the comments. In a study by Breu et al. [10] investigated bug reports is the interaction between users and developers, aiming questions posted by users and respective answers by the developer. These conversations can be happened using set of time stamped comments. Several approaches are considered to generate the summaries of a software artifact. Lotufo et al. [11] examined an unsupervised learning approach for bug extractive summarization of bug reports. Using this approach sentences is ranked based on positional features, such as whether sentences related to frequently discussed a topic or similar to the bug report title and description. The work presented in this paper is to produce meaningful and useful summaries of bug reports and evaluate the summaries using manual survey.

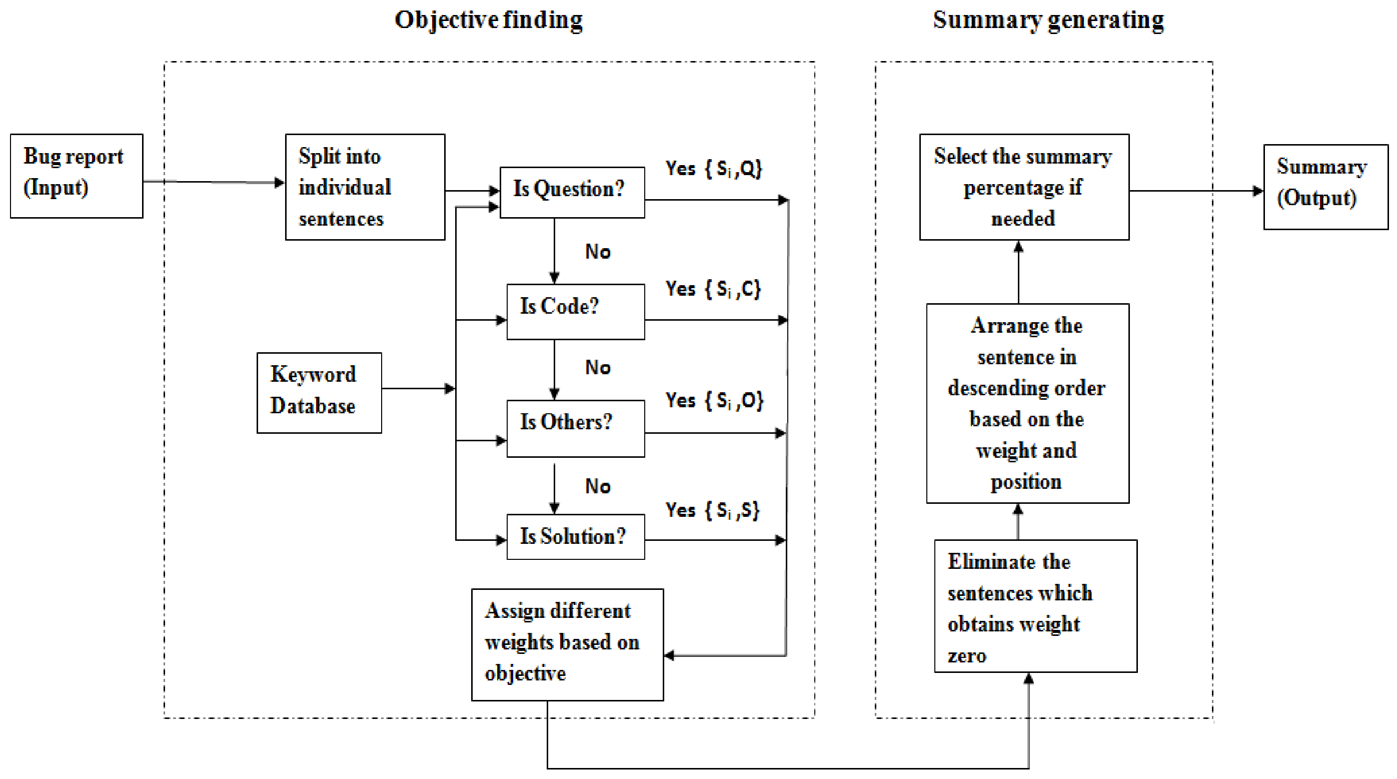

Objective finding
Summary generating
Bug report (Input)
Split into individual sentences
Is Question?
Yes {Si,Q}
No
Is Code?
Yes {Si,C}
No
Keyword Database
Is Others?
Yes {Si,O}
No
Is Solution?
Yes {Si,S}
Assign different weights based on objective
Select the summary percentage if needed
Summary (Output)
Arrange the sentence in descending order based on the weight and position
Eliminate the sentences which obtains weight zero

Fig.2. An Objective Based Approach to Bug Report Summarization.

III. Bug Report Summarization

Fig.2. outlines an objective based approach which contains two phases namely objective finding phase and summary generating phase. These two phases are described as follows:

3.1. Objective Finding Phase

As discussed in section 1, only series of time stamped comments taken into account to generate the summary. In this phase, the bug report is split into individual sentences and we find the objective of each sentence i.e. what kind of sentence it is? The weight to each sentence assigned based on the objective. Conversations in bug reports are classified into four categories based on the information given in [3] and here the weight to each sentence was calculated based on trained data. The four categories of sentence as follows:

- ***Question Sentences:*** These sentences describe the problem being faced and reported by the developers and contain words like *what*, *why*, *how* and probably ends with a question mark (?).
- ***Solution Sentences:*** These kind of sentences describe answer or solution to the problems which are been placed by different developers. These sentences allow users to understand the problem and give their respected solution to solve that problem and typically contain domain specific information or domain specific keywords. Sentences belonging to this group give exact solution or suggestions to solve the problem.
- ***Code Sentences***: These sentences contain code words or phrases or some commands which are domain specific.

- ***Others:*** These are acknowledgement or greeting sentences, for example, *'Hi', 'Hello', 'thank you very much'*, etc.

The following are the few examples that show sentence classification for sample bug reports. Here the label S_i specifies the sentence number and words in a bold font which represents an objective of the sentence.

- S_1: How can we avoid data truncation issue faced by the customer? (**Question**)
- S_2: System.out.println(" enter the number"); (**Code**)
- S_3: Thank you for your support. (**Others**)
- S_4: To solve this problem, add MySql connector and place it in your project library folder. (**Solution**)

3.2. Summary Generating Phase

After finding the sentence objective, weight is assigned to each sentence based on the category it belongs to and weight can be calculated using below formula:

$$\textit{Weight of the Sentence (Wi)} = \begin{cases} W_i = 0 & \text{if } S_i \in \{Q, C, O\} \\ W_i = Value & \text{if } S_i \in S \mid Value \geq 1 \end{cases}$$

Where S_i indicates selected sentence for calculating the weight and W_i indicates the weight of the sentence. Weight of the sentence varies based on the category, for example, sentence of type solution having more weight than a question. As a part of the research an annotation process was performed
with three jurors and they are requested to select an important sentence. A summary is created by grouping the sentences which are linked by two or more jurors. The obtained summary is called as *Human Extractive Summary* (*HES*). In this process no jurors are interested in *code, question and other* kind of sentences and they gave high priority to solution and then suggestions. Based on this work the sentence which belongs above three categories is simply eliminated by assigning value to variable W_i to 0. The sentences which obtain the score zero are filter out without display in the summary and remaining sentences are arranged in descending order based on their score. The sentence elimination done here was not only for reducing length, but also providing only useful information to the developer. If sentences having the same score are arranged based on their position, it means which sentence existed first in the original bug report is placed first in summary. Fig.3 shows the summary to a bug report which is shown in Fig.1.

IV. Analytical Evaluation

We use four metrics to evaluate our approach, namely Precision, Recall, F-Score and Pyramid Precision. To calculate these metrics, we need to find two parameters: True Positive rate (TP) and Fault Positive rate (FP).

- ***True Positive rate (TP):*** It is the ratio between sentences selected from the HES to sentences in HES.
- ***Fault Positive rate (FP):*** It is the ratio between sentences selected that are not in the HES to sentences in the bug report that are not in HES defined in [3].

The following metrics are used to evaluate the performance of summarization which is done by objective based approach.

- ***Precision:*** It measures how accurate our approach chooses the sentences for summary. It is computed as follows:

  Precision (P) = (TP / (TP+FP))

- ***Recall:*** It measures how many of the sentences in GSS are actually chosen by our approach.

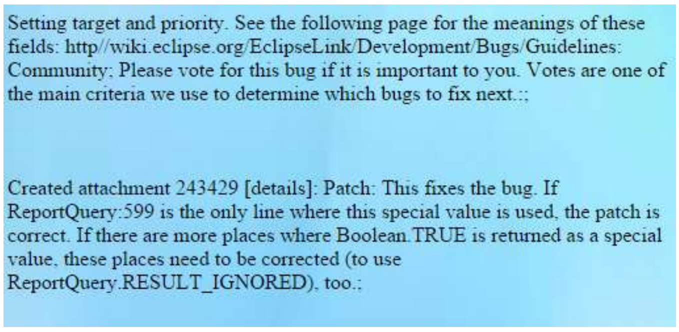
  Recall (R) = (TP / (TP+FN))

  Where FN is Fault Negative rate

- ***F-Score:*** It is a harmonic mean of precision and recall. It is defined as

  F-score= 2*((P*R) / (P+R))

- **Pyramid Precision:** This metric is used when multiple reference summaries are available and Giuseppe Carenini et al. [12] suggested a metric which is used in this approach. Pyramid precision is the ratio of the total number of times the sentences were linked by jurors for a given length to the greatest possible sum for that summary length.


Setting target and priority. See the following page for the meanings of these fields: http//wiki.eclipse.org/EclipseLink/Development/Bugs/Guidelines: Community: Please vote for this bug if it is important to you. Votes are one of the main criteria we use to determine which bugs to fix next.:;

Created attachment 243429 [details]: Patch: This fixes the bug. If ReportQuery:599 is the only line where this special value is used, the patch is correct. If there are more places where Boolean.TRUE is returned as a special value, these places need to be corrected (to use ReportQuery.RESULT_IGNORED), too.;

Fig.3. Summary generated by our system using Objective Based Approach.

Table 1: Evaluation Measures

Approaches	Precision	Recall	F-Score	Pyramid Precision
Objective Based	0.59	0.38	0.46	0.62
BRC	0.57	0.35	0.40	0.66

EC	0.43	0.30	0.32	0.55
EMC	0.47	0.23	0.29	0.54

Table 1 represents Precision, Recall, F-Score and Pyramid Precision values of existed and present approaches. Even though Bug Report Corpus (BRC) classifier based approach produces high pyramid precision, the objective based approach achieves better Precision, Recall, F-Score value. So these values reflect that our objective based approach better suits for summarizing of bug reports.

V. Experimental Evaluation

5.1. Time to Complete

An experimental evaluation was performed to know how much time will take to read and understand the bug under two conditions: *original bug reports* and *summaries*. Four inexperienced participants are requested to resolve the bug by going through four different bug reports which was selected at random. First, we requested to perform the task by reading original bug reports and then summaries. In an average, 10 minutes of time are enough to complete the task. So, ten minutes were assigned to all four participants. For first four bug reports, the Fig.4 represents the amount of time to complete the task for each bug report under the two conditions by four users. In this process participants are taken an average of 60% less time to solve the bug using summaries when compared to original bug reports.

5.2. Participant satisfaction

Three out of four participants are satisfied and preferred to work with summaries. They give good feedback about summaries generated by Objective Based Approach. Participants are feeling better for working with summaries rather than original bug reports and they completed the task within a short time and accurately. In this evaluation, the objective based approach achieves 75% of user satisfaction.

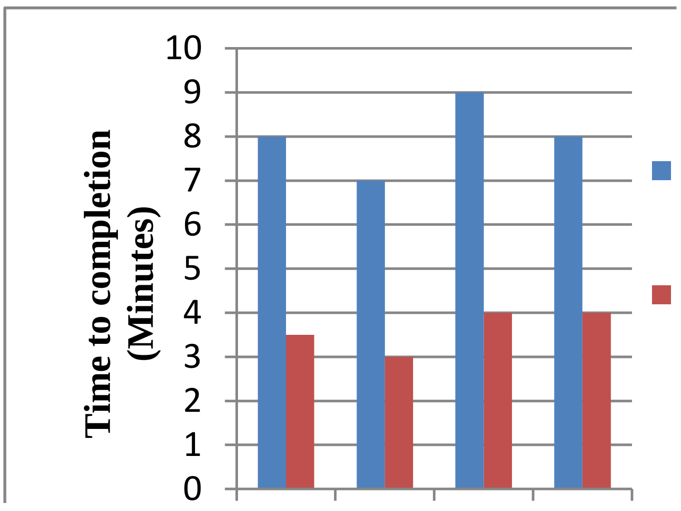


Fig. 4. The average time to complete the task using original bug reports and summaries

VI. Conclusions And Future Work

The Objective Based Approach, which is suggested in this paper is helpful to bug report summarization and mostly applicable to different bug management systems. It also allows users to read and resolve the bugs in less time. Analytical evaluation using different metrics shows better values for this approach when compared with existing approaches. This approach also reduces the noise by eliminating unnecessary informations while producing the summary. Experimental evaluation performed by different participants shows that summaries produced by this approach take less time to resolve bugs when compare to the original bug report. The usability is also improved by providing a way to choose expected summary percentage by users.

One direction of future work is to generate the abstractive summary by analyzing the multiple sentence objectives. Another direction of future work is to apply this approach to generate summaries for different artifacts. For example, generate summary to the requirements document and conversations of a feedback system.

References

[1] D. Cubranic and G.C. Murphy, "Hipikat: Recommending Pertinent Software Development Artifacts", Proc. 25th Int'l Conf. Software Eng. (ICSE'03), pp. 408-418,2003.

[2] Artifacts: A Case Study of Bug Reports", Proc. 32nd Int'l Conf. Software Eng. (ICSE '10), vol. 1, pp. 505-514, 2010.

[3] S Rastkar, Gail C Murphy and Gabriel Murray, "Automatic Summarization of Bug Reports", Proc. IEEE transactions on Software Eng., Vol.40, no.4, 2014.

[4] E.Hovy and C.Y.Lin, "Automated Text Suummarization in SUMMARIST", Pages 18-94. MIT Press, 1999.

[5] X. Zhu and G. Penn, "Summarization of Spontaneous Conversations", Proc. Ninth Int'l Conf. Spoken Language Processing (Interspeech '06-ICSLP), pp. 1531-1534, 2006.

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

[7] Gabriel murray, Giuseppe Carenini and Raymond Ng, “Generating and Validating Abstracts of Meeting Conversations: a User Study”, Proc. Sixth Int’l Natural Language Generation Conf. (INLG ’10)’, pp. 105-113, 2010.

[8] Alok Ranjan Pal, Projjwal Kumar Maiti and Diganta saha, “An Approach To Automatic Text Summarization Using Simplified Lesk Algorithm And Wordnet”, International Journal of Control Theory and Computer Modeling(IJCTCM), Vol.3, no.4/5, 2013.

[9] X.Wang, L.Zhang, T.Xie, J.Anvik and J.Sun, “An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information”, Proc. 30th Int’l Conf. Software Eng. (ICSE ’08), pp. 461-470, 2008.

[10] S. Breu, R. Premraj, J. Sillito, and T. Zimmerman, “ Information needs in Bug reports: Improving Cooperation between Developers and users”, Proc. ACM Conf. Computer Supported Cooperative Work(CSCW ’10), pp. 301-310, 2010.

[11] R. Lotufo, Z.Malik and K.Czarnecki, “ Modelling the ‘Hurried’ Bug Report Reading Process to Summarize Bug Reports”, Proc. IEEE 28th Int’l Conf. Software maintenance(ICSM ’12), pp. 430-439, 2012.

[12] Giuseppe Carenini, Raymond T. Ng, and Xiaodong Zhou, “Summarizing emails with conversational cohesion and subjectivity”, ACL ’08, pages 353–361, 2008.

Selected Paper from International Conference on Computing (NECICC-2k15)