Bug Report Summarization: An Evaluation of Ranking Techniques

Isabella Ferreira*, Elder Cirilo†, Vinícius Vieira†, Fernando Mourão†
* Informatics Department, PUC-Rio
Rio de Janeiro, Brazil
Email: isabellavieira57@gmail.com
† Department of Computer Science, Federal University of São João del-Rei
São João del-Rei, Brazil
Email: {elder, vfvieira, fhmourao}@ufsj.edu.br

***Abstract*—Bug reports are regularly consulted software artifacts, especially, because they contain valuable information for many change management tasks. Developers consult them whenever they need to know already reported problems or have to investigate previous bug solutions. This activity, however, consumes a substantial amount of time once bug reports content might achieve dozens of comments and thousands of sentences. One recommended and massively applied solution to prevent developers to read the entire bug report is to summarize the whole conversation in a few sentences. Summaries ideally give to developers an overview of the current status of the bug and the reasons, highlighting the result of each proposed solution, for which environments, which solutions are most appropriated, and the necessary information to reproduce the bug. This strategy intends to minimize the time a developer would spend in maintenance tasks. However, investigations demonstrate that summaries do not meet the expectations of developers and, in practice, they still read the entry line of discussion. To circumvent this drawback, we propose a summary based on comments, instead of the ones based on isolated sentences, as proposed by previous works. We argue that a ranking of the most relevant comments would enable developers to find more appropriate information. Empirical results corroborate with our arguments and show that summaries generated by traditional ranking algorithms are accurate with respect to developers expected information when compared to reference summaries created manually.**

I. Introduction

A bug occurs as a result of an error, defect or mistake, generating incorrect behavior in the software. In face of unexpected software behavior, a developer or a user reports it in appropriated software artifacts (e.g., bug reports). Bug reports, over a period of time, accumulate valuable information about the described problems, being frequently consulted by developers. They refer to bug reports to see whether a similar problem was resolved in the past and how, to detect duplicated bug reports, to gather the necessary information to reproduce the bug, and to consult stack traces and test cases.

Bug reports, however, are not built to be easily read [2]. They vary in quality of content [5] and in most cases, are only informal conversation and opinions about the failure to be fixed or about an issue to be resolved. In each bug report, developers, and sometimes users, collaborate by posting their contributions as comments. After every new comment, there is a chance of the conversation to become more interlaced and to go in many directions, forcing the readers to keep control of additional contexts on their own. Therefore, since a large number of bug reports can be opened, and each one might contain tens of comments and thousands of sentences, keeping track of the knowledge built on bug reports is not easy, and, in general, consumes a substantial amount of time.

One recommended solution to prevent developers of reading the entire bug report is to create what is known as Extractive Summaries – collection of sentences extracted from the original bug report. Extractive summaries ideally give to developers an overview of the current status of the bug and the reasons, highlighting the result of each proposed solution, and providing the necessary information to reproduce the bug. This strategy intends to minimize the time a developer would spend in maintenance tasks. In [3], extractive summaries are created by trained classifiers via a corpus of reference bug reports. The resulting summary is built by selecting the sentences considered relevant. Since the quality of the resulting summary depends on the quality of the corpus [2] [4], a common weakness of supervised approaches, recently, in [2] [4], the authors proposed unsupervised solutions to summarize bug reports. These summarizers do not require initial setup and, as demonstrated by [2], are more attractive because can be successfully applied in any software development project, specially in open source projects available in collaborative platforms (eg. GitHub), like as jQuery, Angular.js and Bootstrap.

Recent works demonstrate, however, that sentence-based extractive summaries do not meet the expectations of general readers and, in practice, they fail on reducing the software maintenance workload. Lotufo *et. al* [2] recognize the limitations of summaries based on chucked sentences, and argue that they might not be the ideal solution, once developers are not able to directly find the most appropriated information [5]. To circumvent the missing information on bug report summaries, we propose a novel summarization strategy based on comments, instead of those based on isolated sentences, as proposed by previous works [3] [2] [4]. We argue that ranking the most relevant comments would enable developers to find more appropriate information. So, considering the bug reported as a query and the comments as a collection of documents, the problem of summarizing a bug report can be modeled as a standard ranking task in Information Retrieval. We propose to resolve the sentence summarization drawbacks as a ranking problem, in which comments are mainly ranked with respect to their relevance to a given bug report description.

In this paper, we apply four techniques: Cosine Similarity, Euclidean Distance, PageRank and Louvain community detection to the problem of delivering to developers the most relevant comments. Through all evaluated projects, the results show that Cosine Similarity and PageRank are able to produce at least average summaries. These techniques achieved an average Precision of 0.40. PageRank, a more sophisticated technique, offers good results in general. However, it might suffer to result in precise summaries for poor or small bug reports. Cosine Similarity is a safer technique to use once it performed uniformly along all projects. Euclidian Distance and Louvain community detection, on the other hand, seems to be not appropriate to be applied in the context of bug report summarization. Based on the evaluation, we also observed that the ranking of the most relevant comments would enable developers to find more appropriate information than when they consult sentence-based summaries. Euclidean Distance technique, for example, generated summaries with almost the same quality as pointed out by developers. Therefore, empirical results corroborate with our arguments, and show that summaries generated by traditional ranking algorithms are accurate with respect to developers expected information.

The remaining of the paper is organized as follows. In Section 2, we give a brief overview of the problem with sentence-based summaries and related works. In Section 3, we present how the problem of ranking comments was modeled. Section 4 describes the experimental studies conducted and discusses the results. In Section 5, we present some threats to validity. Finally, Section 6 concludes the paper and presents some directions for future works.

II. Problems with Bug Reports

Currently, there is a major direction for automatic summarization of textual bug reports: to provide to developers an Extractive Summary containing about 25% to 30% of sentences from the original bug report. These sentences are further consulted by developers to get an overview of: (i) current status of the bug and the reasons; (ii) proposed solutions; (iii) necessary steps to reproduce the bug.

In [3], the authors suggest the use of summarization techniques based on classifiers to automatically reduce the effort of reading and understanding bug reports. The authors mentioned that there is a huge similarity between conversations in bug reports and other types of conversation, such as email threads, where supervised techniques have been applied successfully. The authors also derived a bug report summary from a trained classifier based on a corpus of bug reports summaries created manually. The authors reported a precision of 63% when the classifier was trained on bug report from the same subject. When trained in a corpus of different subject, the classifier achieved a 54% precision.

Since the quality of the resulting summary is dependent on the quality of the corpus, Lotufo et al. [2] and Mani et al. [4] proposed approaches where the relevant sentences are chosen by unsupervised classifiers. The proposal of Lotufo et al. [2], for example, do not require any initial setup and manual creation of a basic corpus for the classifier training. The summarization is based on a hypothetical model of how one “in a hurry” reads a bug report. Therefore, the authors indicate that the reader, in most cases, will focus only on the most important decisions. To classify sentences as relevant and not relevant, the authors suggest three hypotheses. The most relevant sentences are: (i) the ones that have frequently discussed topics, (ii) the ones evaluated by other sentences and (iii) the ones that focus on the title and description of the bug in question. Each sentence is classified by their probability of being read, and finally, the summary consists of the most likely, or the most relevant sentences.

Although the proposed works successfully generate summaries, interviews conducted with developers [2] indicate that 40% of the sentences recognized as relevant by automatic classifiers are considered irrelevant by developers. That is, the summaries fail in abstract the most important kind of information: (i) solutions to the bug; (ii) suggestions and evaluations about the proposed solutions; (iii) steps to reproduce the bug. These findings were corroborated by studies conducted by Bettenburg et al. [5]. In a survey conducted with developers and reporters about the content of bug reports, the steps to reproduce the bug, observed and expected behavior, stack traces and test cases were rated as the most important items in a bug report, and such information usually is not present in extractive summaries [2].

Therefore, summaries based on chucked sentences might not be the best solution, and, yet 46% of developers prefer to face the tedious reading of the entire bug report. To solve the presented shortcomings, that is, in order to present information that developers expect to find in the summaries, we propose summarization based on comments, instead of the ones based on isolated sentences. A ranking of the most relevant comments would enable developers to find more appropriate information, such as, solutions to the bug suggestions and evaluations about the proposed solutions, steps to reproduce the bug, stack traces, code samples and so on. We propose and evaluate the solution to resolve the sentence summarization drawbacks as a ranking problem, in which comments are mainly ranked with respect to their relevance to a given bug report description. Next, we present how the problem of ranking comments was modeled on four different techniques: Cosine Similarity, Euclidean Distance, PageRank and Louvain community detection; and discuss the results of each technique regarding to the quality of the summaries produced.

III. Summarizing Bug Reports

The bug report summarization process works as follows (see Figure 1). For a collection of bug reports that need to be summarized, we first pass it through a text preprocessing step. Here, we represent the bug reports in a structured format

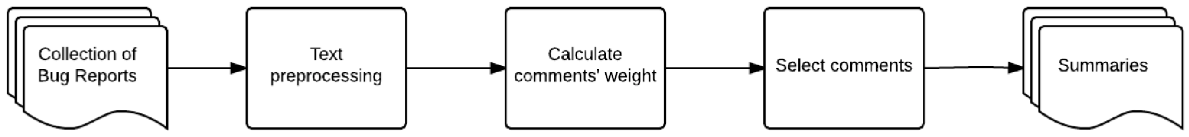

Fig. 1. Process to create bug reports' summaries

that preserves the general characteristics of the text. Next, each preprocessed comment is weighted. We apply 3 different techniques for evaluating comments' weight: cosine similarity; euclidian distance; and comments' conversation characteristics. Finally, comments are ranked by relevance and the most relevant comments are seleted to compose the summary. The next sections present in details the summarization process.

A. Text Preprocessing

In order to rank comments, we need to preprocess the bug reports. The preprocessing of texts aims to structure a text suitable for knowledge extraction. There are several processing techniques that select the most significant terms and represent the text in a structured format that preserves the main characteristics of the information [8]. We choose to proceed by separating each comment of the bug report in terms (or tokens). Next, we convert all characters to lowercase, eliminate digits and punctuation marks, remove numbers, URLs, special characters, stopwords, and therefore we stem the tokens by using the Snowball stemmer [2].

B. Computation of Comments' Weight

Bug reports are the result of a conversation or messages of several people ordered in sequence, where each message is composed of structured and unstructured information. In general, bug reports have the following characteristics: (i) the first comment is always the bug description; (ii) the subsequent discussion is around how to solve the bug and it is usually linked to previous experience of the developer; (iii) participants can aprove or disaprove (evaluates) the ideas on how to solve the bug (represented by +1 or -1); (iv) "pointers" to another bug report resolving similar problems can be included; and (v) "pointer" to commits and/or pull-requests (usually associated with the comment of a particular user) could also exist; finally (vi) the discussion can also involves another matter outside the scope of the bug in question. We chose three different strategies to compute comments' weight based on the probability of a reader focuses his attention on that conversation characteristics, as following:

- Cosine Similarity: calculates the cosine similarity between comments' content and the title and bug report description.
- Euclidian Distance: calculates the euclidean distance between comments' content and the title and bug report description.
- Conversation Characteristics: as cosine similarity and Euclidian distance measures how much two comments share a topic, here we also propose to quantify the relevance of a comment according to the number of associated events (number of evaluative comments, number of commits, pull requests).

C. Selection of Comments using Ranking Techniques

In this section we give a brief overview of each of the four ranking techniques we used in our study: Cosine Similarity, Euclidian Distance, PageRank and Louvain's Algorithm. Cosine Similarity and Euclidian Distance are the most used techniques in the literature to measure distance and similarity among objects. PageRank is the state-of-art ranking algorithm, and Louvain's Algorithm is the state-of-art algorithm to find communities in a graph [13].

1) ***Ranking as Distance and Similarity Measure****:* The distance and similarity measure reflect the degree of closeness or separation of the topic shared by target objects [9]. As we consider the title and the description of the bug as reference elements, we also consider the more relevant comments the ones which has similar topics to the description and to the title of the bug report.

To calculate Cosine Similarity and Euclidian Distance, we first need to represent the bug report D as a m-dimensional vector $\overrightarrow{t_d}$. Being $T = \{t_1, t_2, ..., t_m\}$ a set of distinct terms occurring in a comment $d \in D = \{d_1, d_2, ..., d_n\}$; we have that *tf(d,t)* denotes the frequency of term t in a comment d [9]. Then, the vector representation of a comment d is:

$$\overrightarrow{d_i} = (tf(d, t_1), tf(d, t_2), ..., tf(d, t_m)), \tag{1}$$

Although more frequent words are assumed to be more important, this is not usually the case. We use the $tfidf$ weighting (term frequency inverse document frequency) to indeed emphasize the most important words. Thus, $tfidf$ is represented by:

$$tfidf(t, d) = n_{t,d} log \frac{N}{n_t} \tag{2}$$

where $n_{t,d}$ is the number of times the term t occurs in d, N is the total number of comments and n_t is the number of comments that has the term t. After calculating the $tfidf$ of each term in a bug report, we apply the Non-negative matrix factorization (NMF) method over $tfidf$ in order to reduce its dimensional space.

The NMF is a vector space method to obtain a representation of information using non-negativity constraints. These constraints can lead to a parts-based representation because they allow only additive, not subtractive, combination of the original information [11]. NMF refers to the following computation. Given a $m \times n$ matrix A with only non-negative entries, and given an integer parameter k such that $k \leq min(m, n)$, find two matrices $W \in R\ m \times k$ and $H \in R\ k \times n$ such that $A \approx WH$, and such that W and H both have non-negative entries [10] [19]. We chose to use NMF instead of other methods for reducing dimensionality, such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA), because the negative components contradict our reality, since the term-frequencies are non-negative.

Now, we calculate the Cosine Similarity (Section III-C1) and the Euclidean Distance (Section III-C1) among all comments with the title and description of the bug. At the end, we sort in descending order the values of the Cosine Similarity, and in ascending order the values of Euclidian Distance.

Cosine Similarity: When comments are represented as term vectors, the similarity of two comments corresponds to the correlation between the vectors. This is quantified as the cosine of the angle between vectors [9]. Cosine similarity, which has shown consistent results in measuring the similarity of content [2], can be defined as

$$cosine - similarity(d_i, d_j) = \frac{d_i \cdot d_j}{\mid d_i \mid\mid d_j \mid}, \tag{3}$$

where d_i corresponds to the representation of a comment indexed by i and d_j corresponds to the representation of a comment indexed by j.

When applied to the original term space, if the value of cosine similarity is 0, the angle between d_i and d_j is 90^o, i.e., the comments do not share any term. On the other hand, if the value of the cosine similarity is close to 1, the angle between d_i and d_j is close to 0^o, indicating that comments share terms and are similar.

Euclidean Distance: Euclidean distance is a standard metric for geometrical problems. It is the ordinary distance between two points and can be easily measured with a ruler in two- or three-dimensional space [9]. Measuring distance among text comments, given two comments d_i and d_j represented by their term vectors $\overrightarrow{t_i}$ and $\overrightarrow{t_j}$, respectively, the Euclidian distance between two comments can be defined as

$$euclidean - distance(d_i, d_j) = (\sum_{t=1}^{m} |w_{t,i} - w_{t,j}|^2)^{\frac{1}{2}} \tag{4}$$

where the term set is $T = \{t_1, t_2, ..., t_m\}$, $w_{t,i} = tf - idf(d_i, t)$ and $w_{t,j} = tf - idf(d_j, t)$.

*2) **Ranking with PageRank:*** PageRank was developed to rank Web pages by relevance [14]. The original algorithm estimates the relevance of a Web page as the probability of a user reaching that page surfing ramdonly on the Web following hyperlinks from one page to another [2] [18]. Considering the Web example, PageRank takes as input a graph G, where Web pages are nodes and hyperlinks from one page to another are directed edges. Then, PageRank calculates the Markov chain for the random surfer model and outputs the probability of a user eventually reaching the Web page i after a large number of clicks [2] [17].

$$M_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \alpha cos\_similarity(s_i, s_0) + \beta \sum events/totalEvents & \text{if } i = 0 \vee j = 0, \\ \gamma cos\_similarity(s_i, s_0) + & \\ \delta \sum cos\_similarity(s_i, s_j) / \sum similarities + & \\ \lambda \sum events/totalEvents & otherwise. \end{cases} \tag{5}$$

As the random surfer model is similar to the way a user reads a bug report [2], we propose to apply PageRank to summarize bug reports by ranking the most important comments. In our case, we have a graph G in which comments are represented by nodes, and the links are represented by weighted edges. The edges are weighted in accordance with the Conversation Characteristics strategy. They are weighted by the sum of the number of referenced commits, the number of references to pull requests, commits and/or other events (close, open) and the number of likes (+1) a bug report has, as in equation 5. The graph is represented as a matrix M $n \times n$, where n is the number of comments of the bug report. However, we do not have any guarantee that M is irreducible and aperiodic. In order to guarantee the convergence of the matrix M, we choose a threshold to turn the matrix into a sparse matrix. We then transform the matrix M into a column stochastic matrix, so that the convergence of the stochastic matrix is guaranteed. Finally, we run PageRank and find the probabilities for each comment. The comments that have higher probability are considered as the most relevant comments.

*3) **Ranking as Community Detection:*** To rank comments by applying community detection algorithm, we first create a graph G such as described in Section III-C2. In G, comments are nodes and edges are also weighted in accordance with the Conversation Characteristics strategy, that is, they are weighted considering the sum of the number of referenced commits, the number of references to pull requests, commits and/or other events (close, open) and the number of likes (+1) a bug report has, as decribed in equation 5. We represent the graph as a matrix M of dimensions $n \times n$, where n is the number of comments of the bug report. Then, we apply the NMF method to it, in order to reduce the dimensionality of the matrix, creating a new matrix R of size $n \times k$, where k is an empirical number of columns of the new matrix. The matrix R therefore is transformed into a sparse matrix and the Louvain method (see next) is applied to detect communities in the graph. Afterwards, we identify the most important comments in the community in which the title and the description are included. Finally, we calculate the eigenvector centrality (see next) and select the most central comments as a result.

Louvain Method: As described in [13], Louvain method is one of the state-of-art methods for modularity maximization for detecting community structure in networks. The method is simple, efficit and easy to implement and it is well-suited for analyzing large weighted networks. Louvain method is based on local information and seeks to optimize local communities until the global modularity cannot be improved. It is based on two simple steps: (i) each node is assigned to a community chosen in order to maximize the local modularity Q; (ii) the gain derived from moving a node i into a community C is calculated and nodes are joined in order to belong to the same community. A new network, in which the communities are represented by nodes, is built and these steps are repeated until the maximum modularity is achieved and a community hierarchy is produced [13] [16].

Eigenvector Centrality: Eigenvector centrality gives high importance to a vertex based on the relationship with its neighbors, that is, if a vertex v_i is connected only to another vertex v_j, neighbors of v_j can be important, and therefore the vertex v_i will also be important, getting a high eigenvector centrality [12]. The meaning of the eigenvector centrality as a measure is because the eigenvector defines the most central vertex which is connected to the other which in turn also establish relations with vertices that are in central position, and so on. The eigenvector centrality of a vertex is a linear combination of the centrality of vertices connected with them ([12]). In general, the algorithm determines the eigenvalue of the largest absolute value of a matrix and its corresponding eigenvector.

IV. Evaluation

In this section, we present an empirical evaluation of the ranking techniques with respect to a reference summary created manually by software developers based on a predefined corpus. More specifically, we want to characterize ranking techniques in the context of bug report summary production, and answer the following research questions:

- **RQ1 - Feasibility of ranking techniques for summarizing**: To what extent ranking techniques produce relevant comment-based summaries?
- **RQ2 - Quality of the produced summaries**: How good are the ranking techniques in suggesting comments classified as Steps to reproduce the bug, Solution or workarounds, Evaluations about the discussed topics.

A. Experimental Setup

1) Selection of Bug Reports: In order to evaluate the four ranking techniques, we chose 50 bug reports from the following open source projects mostly starred and forked on GitHub: *bootstrap*[1], *angular.js*[2], and *jquery* [3].

The 50 bug reports were chosen considering that all of them follow the criteria: (i) bug reports that contain mostly natural conversation content with no structured information such as patches, stack traces and source code; (ii) bug reports that have more than 3 people participating on the conversation; (iii) bug reports that have closed status; (iv) bug reports about suggestion of implementation of new features or bugs; (v) bug reports that have between 20 and 45 comments. From the 50 chosen bug reports, we picked, randomly, 15 bug reports. With the chosen 15 bug reports, we asked some developers to manually create 15 bug report summaries (5 bug reports summary from each software project). Table I shows the mean of words, sentences and comments for each dataset. Next, we briefly describe each open source project.

a) Bootstrap: Bootstrap is a free and open-source collection of tools for creating websites and Web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications. Bootstrap is the most starred project on GitHub with over 94k stars and more than 40k forks.

b) AngularJS: AngularJS is an open-source web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model-view-controller (MVC) and model-view-view-model (MVVM) architectures, along with components commonly used in rich Internet applications. It has over 48k stars and more than 22k forks on GitHub.

c) jQuery: jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery is the most popular JavaScript library in use today. The project has over 38k stars and over 10k forks on GitHub.

TABLE I
Mean of words, sentences and comments for each dataset

Project	#words	#sentences	#comments
Angular	2834 ($\pm$644.42)	110.6 ($\pm$31.97)	36.4 ($\pm$5.59)
jQuery	1829.4 ($\pm$472.51)	86 ($\pm$22.03)	30.4 ($\pm$8.90)
Bootstrap	1420.6 ($\pm$459.14)	75.4 ($\pm$23.9)	35 ($\pm$6.04)

2) Selection of Participants: We chose 9 participants, with software development experience, who speak English fluently,

[1]https://github.com/twbs/bootstrap
[2]https://github.com/angular/angular.js
[3]https://github.com/jquery/jquery

to create manually 15 bug report summaries. Before proceeding with the summarization task, each participant filled out a form about previous experience in software development as well as experience in the software projects. As a result, 75% of the participants has experience with AngularJS, 87.5% has experience with Bootstrap and 100% of the participants has experience with jQuery. Further, 62.5% has experience with software development in industry, 75% has experience with research projects and 37.5% of the participants has experience in industry and research projects. The participants also have more than 4 years of experience in software development.

3) Oracle creation: We randomly assigned to each participant 5 bug reports from 3 projects (Angular, jQuery and Bootstrap). Each bug report has about 30 comments and was evaluated by three different participants. To complete the summarization task, firstly the participants were asked to read the bug report in order to obtain a good understanding of what the bug report is about. Then, they were asked to reread the bug reports and rate each comment according to the importance of the information contained therein. The importance levels were defined according to the five-point Likert scale ranging from (1) not important at all to (5) extremely important. We have defined that a comment is important if the topic discussed in that comment is similar to the topic of the title and the description of the bug.

The material was distributed to participants as forms in Google Forms. In the forms, each question represented one comment from the bug report. And, associated to each question, singular options where the participants could categorize each comment in the scale listed above. Each participant had 4 days to generate the summaries. No one had contact with the other participants, and no one knew which bug report each participant was evaluating. In order to mitigate the problem of having different evaluations for each comment, an odd number of participants were assigned to each bug report. Therefore, to define the final importance of each comment, we selected the importance by prioritizing the rating that was assigned more frequently. This is a technique called by Majority Vote. Comments with different evaluations were discarded from the comments' list. Further, we sorted the comments from the 'extremely important' to 'not important at all'. At the end, to produce the reference summaries (i.e., the oracle used in our evaluation), we picked the top 10 comments and asked other 4 developers to order these comments from (1) most important to (10) not important at all.

4) Measures of Summarization Effectiveness: In order to evaluate the effectiveness of each ranking technique (Cosine Similarity, Euclidian Distance, PageRank and Louvain), considering the Oracle created, we used the standard evaluation measures of precision, recall and f-score. Precision measures how accurate the predictions are [4] [20]. Recall measures the ability of the algorithm to select the sentences of the summary [4] [20]. As there is a tradeoff between precision and recall, we used f-score to calculate the mean. Equations (6) (7) and (8) describe the calculation of the measures, on which #hits_algorithm is the number of hits in the top 10 ranked comments of the algorithm compared to the top 10 comments of the oracle, #ranking is the number of comments in the algorithm's ranking and #oracle is the number of comments in the oracle.

We used the Spearman's rank correlation coefficient to check the correlation between the ranking generated by each evaluated technique and the Oracle ranking. Spearman correlation computes agreement between two rankings. Two rankings can be opposite (value -1), unrelated (value 0) or perfectly matched (value 1) [5] [6]. Furthermore, we used the Modularity to measure how well the data were distributed into clusters.

$$Precision = \frac{\#hits\_algorithm}{\#ranking} \tag{6}$$

$$Recall = \frac{\#hits\_algorithm}{\#oracle} \tag{7}$$

$$F - score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{8}$$

B. Results and Discussion

We generated summaries for all bug reports by picking the top 10 comments returned. Table II presents the average precision for Cosine Similarity, Euclidean Distance and PageRank. Table III shows the precision, recall and f-score for Louvain method. We did not calculate the recall and f-score for Cosine Similarity, Euclidean Distance and PageRank because the size of the oracle and the summary are both 10 comments. Thus, the value of precision, recall and f-score would be the same. For Louvain method we select all the comments in the cluster in which the title and the description are included. In this case, the summary, in some cases, does not contain exactly 10 comments, so we might observe a variance in precision, recall and f-score. Accordingly, Tables IV, V and VI presents the Sperarman's Rank correlation results for all algorithms and datasets.

TABLE II
AVERAGE PRECISION FOR COSINE SIMILARITY, EUCLIDEAN DISTANCE AND PAGERANK

Project	Average Precision		
	Cosine Similarity	Euclidian Distance	PageRank
Angular	0.42 (±0.18)	0.20 (±0.07)	0.46 (±0.08)
jQuery	0.46 (±0.15)	0.38 (±0.22)	0.56 (±0.10)
Bootstrap	0.46 (±0.16)	0.30 (±0.14)	0.40 (±0.10)

TABLE III
AVERAGE PRECISION, RECALL AND F-SCORE FOR LOUVAIN

Project	Precision	Recall	F-score
Angular	0.27 (±0.12)	0.26 (±0.13)	0.26 (±0.13)
jQuery	0.42 (±0.26)	0.32 (±0.26)	0.34 (±0.26)
Bootstrap	0.45 (±0.18)	0.36 (±0.08)	0.38 (±0.08)

TABLE IV
SPEARMAN'S RANK CORRELATION BETWEEN RANKED COMMENTS AND ORACLE FOR ANGULAR

Spearman Correlation - Angular

Bug Report	#2895	#734	#1412	#583	#5160
Cosine Similarity	0.27	-0.05	0.17	0.56	-0.16
Euclidian Distance	0.62	0.06	-0.15	0.28	0.39
PageRank	0.006	-0.10	-0.12	0.006	-0.66
Louvain	0.02	-0.006	0.18	-0.03	0.68

TABLE V
SPEARMAN'S RANK CORRELATION BETWEEN RANKED COMMENTS AND ORACLE FOR JQUERY

Spearman Correlation - jQuery

Bug Report	#2145	#2310	#2199	#1692	#2321
Cosine Similarity	-0.03	0.006	0.35	-0.32	0.18
Euclidian Distance	0.15	-0.27	0.09	-0.56	0.30
PageRank	-0.11	0.17	-0.05	0.33	0.2
Louvain	0.06	-1	-0.03	0.33	0.29

TABLE VI
SPEARMAN'S RANK CORRELATION BETWEEN RANKED COMMENTS AND ORACLE FOR BOOSTRAP

Spearman Correlation - Bootstrap

Bug Report	#341	#1235	#931	#1602	#1997
Cosine Similarity	-0.32	-0.09	-0.11	-0.33	0.04
Euclidian Distance	0.24	-0.26	-0.46	0.23	0.28
PageRank	-0.62	-0.6	-0.11	0.09	0.13
Louvain	0.10	0.16	0.03	0.054	0

1) Feasibility of Ranking Techniques for Summarizing: We observe that Cosine Similarity presented, in general, more consistent results than Euclidean Distance (average precsion > 0.42), given that it only considers the relative frequencies between the words that are important for all documents. Euclidean Distance, in contrast, is influenced by the size of the documents, which is shallow in the case o bug report comments (see Table I). PageRank also presented consistenty results (average precsion > 0.40). The PageRank modeling metodology considers a number of factors that could have contributed to increase the relevance judgment: (i) the similarity of comments with the bug description; (ii) the frequency of discussed topics; and (iii) the relationship with some events, such as pull requests, commits and open discussions. Louvain method do not presented good results due to the fact that this technique depends on a calibrated threshold which, when applied in a non-supervisioned way, produces clusters with low similarity quality. Modularity results explain the low similarity quality of the clusters, once Angular, jQuery and Bootstrap projects had average modularity of 0.08, 0.13 and 0.14, respectively. We conclude from those results that the data was not well distributed among clusters, resulting in low precision. The non-parametric Kruskal-Wallis test, however, supports that there is no statistical difference among the techniques regarding to precision results with p-value < 0.05.

By analyzing the correlation results, we were able to gather more interesting observations. First, although PageRank has slightly more consistenty average precision than the remain techniques, the generated rankings did not present good correlation with the oracle. That is, the PageRank technique considered just few comments in the same order as the oracle. The non-parametric Kruskal-Wallis test, in this case, support that there is statistical difference among the techniques with p-value > 0.05.

The presented results show that most techniques produce relevant summaries. Moreover, we observed that PageRank offers good results in general, but it might suffer to offer precise summaries when the bug report does not offer all the expected information, that is, it is hard to infer the similarity between each comments and the bug description, there are more than one frequently discussed topics, and no events were associated to comments. Cosine Similarity is a safer technique to use once it performed uniformly along all projects. Louvain and Euclidian Distance, on the other side, seems to be not appropriate to be applied in the context of bug report summarization as comments ranking. Based on these results, we are able to answer our RQ1, and conclude that ranking techniques are viable options to produce relevant summaries as a ranking of the most relevant comments.

We also observed in all bug reports that many comments discuss the same topic or provides very similar option about the reported bug. This is another factor we consider impactful. The precision results, like the ones < 0.20, could be achieved because the techniques judge relevant some comments and the developers judge others one, although, they are discussing the same topic or providing the same option about the reported bug. We mean that, even low precision summaries, could be low qualities one. Therefore, next we discuss exactly this point, the quality of the generated summaries.

2) Quality of the Produced Summaries: In order to check if the summaries contain the information a user expect to see in the bug reports summary (as explained in the Section II), we asked two developers to categorized manually each comment in the top 10 of the rankings. They associated each comment to the following categories: (i) steps to reproduce the bug; (ii) solutions or workarounds; (iii) evaluations about the discussed topics (comments that agree with the purpose of the bug report or emphasize the importance of another comment); and (iv) others (status and reason for that status, comments that are off topic). Table VII presents the percentage of comments in each category out of 150 comments for the Cosine Similarity, Euclidean Distance and PageRank, and 130 comments for the Louvain algorithm.

As we can see from Table VII, Cosine Similarity provides more information about steps to reproduce the bugs. Euclidean Distance and Louvain provide more information about topics out off the scope of the bug report. We could observe that Louvain method and Euclidian Distance were not able to differentiate most of the information presented in the bug reports, mainly because comments that talked about off topics were put together with the title and the bug description in the same cluster, resulting in a high number of comments

TABLE VII
QUALITY OF THE SUMMARIES AND THE ORACLE BY CATEGORIES

Category	Oracle	Cosine Similarity	Euclidean Distance	PageRank	Louvain
Steps to reproduce the bug	7.3%	11.33%	7.33%	10.6%	6.92%
Solution or workarounds to the bug	24.6%	20%	22%	20%	16.15%
Evaluation about the discussed topic	38%	33.33%	28%	34.6%	29.23%
Others	30%	35.3%	42.6%	34.6%	47.68%

categorized as others. These techniques categorized 47.68% and 42.6% of the comments as Others, while in the Oracle, about 30% of the comments is part of this category.

The results show that high accurate summarizer (high Precision and Spearman Correlation) has provided summaries with more consistent quality than low accurate ones. The techniques PageRank and Cosine Similarity, were the ones more consistenty, and the ones which the highest number of comments about Steps to Reproduce the Bug and Evaluation about Discussed Topics. Louvain and Euclidian Distance techniques presented the worse precision. So, as was expected, Louvain hits only 6.92% out of 7.3% comments in the category of Steps to Reproduce the Bug, 16.15% out of 24.6% in the category of Solutions and Workarounds, and, 29.23% out of 38% in the category Evaluations about Discussed Topics. These results answer our RQ2. We might conclude that ranking techniques are able to suggest comments that can be classified as Steps to reproduce the bug, Solution or workarounds, Evaluations about the discussed topics.

3) Summaries Size: In terms of size, we have on average summaries: (i) 30% the length of the original bug report for the Angular; (ii) 37% for the jQuery Project; and (iii) 43% for the Bootstrap (see Table VIII). As our purpose is to provide more information to developers in summaries giving to them the 10 most relevant comments, inevitably, we have generated bigger summaries in respect to the ones in the literature. In [1] [2], the authors generate summaries by selecting sentences until the summary reaches a length of around 25% to 30% in the number of words of the original bug reports. However, as we could observe, the ranking of the most relevant comments would enable developers to find more appropriate information than when they consult isolated sentences.

4) Bug Reports' Quality: Based on a deeper observation of all bug reports, we can state that the quality of summaries depends, mainly, on the quality of the bug report. In the Oracle, most of the information was categorized as evaluation about the discussed topic, been only 24.6% of the information classified as solution or workarounds and 7.3% as steps to reproduce the bug. It means that it is almost impossible to automatic summarize and collect the exact information that the developers consider essential because most of the bug report content is not about solutions, suggestions and evaluations or about the steps to reproduce the bug. Even more, it is rarely to find any stack trace or tips to reproduce the reported bug [5].

V. THREATS TO VALIDITY

This section discuss the study constraints. For each category, we list possible threats and the procedure we took to mitigate their risk.

a) Conclusion Validity: The greatest risk is the lack of knowledge about the projects evaluated here. To mitigate this problem, we chose participants who have a more solid knowledge in the area and we chose projects that are up to date and probably participants have some knowledge about them. Another threat is that bug reports are written in English. To solve this problem, we chose participants who have advanced knowledge in English. In addition, the size and the amount of bug reports may be a threat to the conclusion of this study, as this influences the participants' engagement. To mitigate this problem, we chose bug reports on average size (about 30 comments per bug report).

b) Construct Validity: A threat to the validity of construction is the lack of knowledge of participants in the bug reports for those projects, and there was no training session for this. To minimize this problem, participants were asked to read the entire bug report firstly to the understanding of what was being discussed and after that, to categorize each comment. In addition to that, participants' questions were answered when requested. To avoid a bias in the answers, questions about the importance of the comments on the bug reports were not answered. There was no interaction between the participants and no participant knew which report the other ones were evaluating.

c) Internal Validity: A threat to internal validity is the choice of a threshold to increase the sparsity in the comment-comment matrix, and also the number of components for the non-matrix factorization method. Both thresholds are empirical, and there is no methodology to choose both. Both thresholds affects directly the result of the algorithm. Another threat is that bug reports can contain different types of structured information, which are not being treated in our algorithms. To mitigate this problem, there were selected reports that have minimal amount and may be zero of structured information and have mostly context of conversation between more than 3 participants.

d) External Validity: The number of participants and the number of bug reports may not represent the amount of reports and the size of comments that will be summarized in practice. To reduce this risk, we have chosen the GitHub repository that possesses the most important projects of nowadays and

TABLE VIII
AVERAGE OF #WORDS, #SENTENCES AND #COMMENTS FOR THE SUMMARY PRODUCED BY EACH TECHNIQUE

Project	Cosine Similarity			Euclidean Distance		
	#words	#sentences	#comments	#words	#sentences	#comments
Angular	919 (±211.13)	42.2 (±8.07)	10	572.4 (±396.32)	25.4 (±20.00)	10
jQuery	768.2 (±231.89)	35.8 (±13.00)	10	563.2 (±222.82)	27.4 (±8.01)	10
Bootstrap	681.8 (±301.55)	37 (±17.98)	10	297 (±174.12)	15.2 (±9.03)	10

Project	PageRank			Louvain		
	#words	#sentences	#comments	#words	#sentences	#comments
Angular	889.2 (±197.41)	39 (±8.71)	10	612 (±281.94)	29.2 (±1.78)	4.7 (±0.89)
jQuery	603.2 (±208.73)	34.2 (±11.14)	10	571.8 (±447.29)	27.2 (±20.77)	3.9 (±13.27)
Bootstrap	543.8 (±93.93)	30.2 (±6.68)	10	571.2 (±214.03)	31 (±15.57)	4.4 (±2.68)

we chose open source projects that has been widely used in software development.

VI. CONCLUSION AND FUTURE WORK

When unexpected software behavior are observed, developers or users report it through bug reports, which over a period of time, accumulate valuable information about the observed problems. As a consequence, bug reports are regularly consulted software artifacts, especially during many change management tasks. Bug reports, however, are not built to be easily read. They are characterized by informal conversation and opinions about the failure to be fixed or about an issue to be resolved. The reading activity, therefore, might consumes a substantial amount of time. One recommended solution to prevent developers of reading the entire bug report is to create which is known as Extractive Summaries – defined as collection of sentences extracted from the original bug report. In this paper, we propose a novel approach where summaries are based on comments, instead of the ones based on isolated sentences, as proposed by previous works [3] [2] [4].

Empirical results corroborate with our arguments that ranking the most relevant comments would enable developers to find more appropriate information. We were able to observe that summaries generated by traditional ranking algorithms are accurate with respect to developers expected information, when compared to reference summaries created manually, offers relevant summaries in general.

One area of future work is to investigate opportunities to improve the precision of ranking techniques, mainly for those that presented the best performances (PageRank and Cosine Similarity). Another direction for future work is to evaluate the influence of the text preprocessing on the quality of the generated ranking. Finally, appropriated classification techniques could also be applied to enhance the quality of the summaries by delivering to developers groups of comments related to the following categories: (i) solutions to the bug; (ii) suggestions and evaluations about the proposed solutions; (iii) steps to reproduce the bug.

ACKNOWLEDGMENTS

This work is funded by CAPES/Procad (grant # 175956), CNPq (grants # 483425/2013-3, 309884/2012-8 and 460401/2014-9), FAPERJ (E26-102.166/2013) and Federal University of São João del-Rei.

REFERENCES

[1] Rastkar, Sarah and Murphy, Gail C and Murray, Gabriel. Summarizing software artifacts: a case study of bug reports. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. 2010.

[2] Lotufo, Roberto and Malik, Zaki and Czarnecki, Krzysztof. Modelling the 'hurried' bug report reading process to summarize bug reports. 28th IEEE International Conference on Software Maintenance (ICSM). 2012.

[3] Rastkar, Sarah and Murphy, Gail C and Murray, Glen. Automatic summarization of bug reports. IEEE Transactions on Software Engineering. Volume 40, Number 4, Pages 366–380, 2014.

[4] Mani, Senthil and Catherine, Rose and Sinha, Vibha Singhal and Dubey, Avinava. Ausum: approach for unsupervised bug report summarization. Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. 2012.

[5] Bettenburg, Nicolas and Just, Sascha and Schröter, Adrian and Weiss, Cathrin and Premraj, Rahul and Zimmermann, Thomas. What makes a good bug report?. Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. 2008.

[6] Wasserman, Larry. All of statistics: a concise course in statistical inference. Springer Science & Business Media. 2013.

[7] Thung, Ferdian and Lo, David and Jiang, Lingxiao. Automatic defect categorization. 19th Working Conference on Reverse Engineering (WCRE). 2012

[8] Feldman, Ronen and Dagan, Ido. Knowledge Discovery in Textual Databases (KDT). Volume 95, Pages 112–117, 1995.

[9] Huang, Anna. Similarity measures for text document clustering. Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008). 2008

[10] Vavasis, Stephen A. Algorithms and Complexity for Nonnegative Matrix Factorization. Householder Symposium XVII Book of Abstracts. Volume 401, 178, 2008.

[11] Pauca, V Paul and Shahnaz, Farial and Berry, Michael W and Plemmons, Robert J. Text Mining Using Non-Negative Matrix Factorizations. Volume 4, Pages 452 – 456, 2004.

[12] Ruhnau, Britta. Eigenvector centrality - a node-centrality?. Social networks. Elsevier. Volume 22, Number 4,

[13] De Meo, Pasquale and Ferrara, Emilio and Fiumara, Giacomo and Provetti, Alessandro. Generalized louvain method for community detection in large networks. 11th International Conference on Intelligent Systems Design and Applications (ISDA). 2011.

[14] Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry. The PageRank citation ranking: bringing order to the web. Stanford InfoLab. 1999.

[15] Langville, Amy N and Meyer, Carl D. Who's# 1?: the science of rating and ranking. Princeton University Press. 2012

[16] Ankolekar, Anupriya and Sycara, Katia and Herbsleb, James and Kraut, Robert and Welty, Chris. Supporting online problem-solving communities with the semantic web. Proceedings of the 15th international conference on World Wide Web. 2006.

[17] Brin, Sergey and Page, Lawrence. Reprint of: The anatomy of a large-scale hypertextual web search engine. Volume 56, Number 18, Pages 3825–3833. Computer Networks. Elsevier. 2012.

[18] Langville, Amy N and Meyer, Carl D. Google's PageRank and beyond: The science of search engine rankings. Princeton University Press. 2011

[19] Viana Bicalho, Paulo and de Oliveira Cunha, Tiago and Mourao, Jesus and Henrique, Fernando and Lobo Pappa, Gisele and Meira, Wagner. Generating Cohesive Semantic Topics from Latent Factors. Brazilian Conference on Intelligent Systems (BRACIS). 2014.

[20] Jiang, H and Zhang, JX and Ma, HJ and others. Mining authorship characteristics in bug repositories. Sci China Inf Sci. 2015.