Measuring the Semantic Similarity of Comments in Bug Reports

Bogdan Dit, Denys Poshyvanyk, Andrian Marcus

Department of Computer Science
Wayne State University
Detroit Michigan 48202
313 577 5408
<bdit, denys, amarcus>@wayne.edu

Abstract

Bug-tracking systems, such as Bugzilla, contain a large amount of information about software defects, most of it stored in textual, rather than structured form. This information is used not only for locating and fixing the bugs, but also for detecting bug duplicates, triaging incoming bugs, automatically assigning bugs to developers, etc. Given the importance of the textual information in the bug reports, it is desirable that this text is highly coherent, such that the readers can easily understand it.

The paper describes an approach to measuring the textual coherence of the user comments in bug reports. The coherence of bug reports from Eclipse was measured and the results are discussed in the paper.

1. Introduction

A large part of software development and maintenance is spent on locating and fixing bugs. It is common to use in large projects defect reporting and tracking systems, such as Bugzilla[1]. Such systems collect a lot of information about identified defects, most of it in natural text, such as bug descriptions, user comments, etc.

The information provided in these bug reports influences the time it takes to fix the bugs [2, 16] and it can be used to support tasks, such as, impact analysis [3, 4], detection of duplicate bug reports [13, 14], or assigning bug reports to developers [1, 5, 7]. It has been shown that bug reports greatly differ in their quality of information [10, 11, 15]. The proposed quality models ignore the user comments posted in the bug reports. We argue that good bug reports should contain not only good textual descriptions of the problem and properly selected attributes, but also coherent and relevant comments.

[1] http://www.bugzilla.org

In this paper we propose a novel approach to measure the textual coherence of user comments in bug reports. We consider that the textual coherence of user comments affects the comprehensibility of bug reports hence it is important to measure it. Our measuring technique relies on the utilization of Information Retrieval (IR) techniques, which allows for automatic coherence measurement of user comments in large bug repositories. We measured the coherence of bug reports from Eclipse[2] and our preliminary results suggest that the proposed measure correlates with assessments provided by software developers.

2. Background and Motivation

Bug-tracking repositories provide means of communication among geographically distributed developers and teams. The developers can describe and issue new bug reports, comment on existing bug reports, suggest fixes to the bugs, subscribe to e-mail discussions for specific bug reports, etc.

An individual record in a bug-tracking database is referred to as an *issue or bug report*. A typical bug report consists of several components such as *title* (or short summary); *attributes* or pre-defined fields such as bug report id number, creation date, reporter, product, component, operating system, version, priority, severity, e-mail addresses of developers on the mailing list for the bug, etc.; *long description* and *comments*, which are posted by developers.

Bugzilla's published usage rules specify the following about writing comments: "*If you are changing the fields on a bug, only comment if either you have something pertinent to say, or Bugzilla requires it. Otherwise, you may spam people unnecessarily with bug mail.*"[3]

Each project usually defines its own guidelines on how to post comments in the bug reports. For example,

[2] http://www.ecplise.org
[3] http://www.bugzilla.org/docs/2.18/html/hintsandtips.html

Mozilla developers, are encouraged to write such comments: *"If you make a lot of useful comments to someone's bugs they may come to trust your judgment and ask you to go ahead and make the changes yourself, ..."*[4].

A common side effect across projects is that these comments become a source of discussion among developers. Developers often tend to comment on each other's comments, rather than on the changes, and the discussion often degenerates and loses coherence.

3. Measuring the Coherence of Bug Reports

Our approach to measuring the textual coherence of bug reports is based on the premise that the comments in bug reports should relate to the problem described in the bug report and should form a *coherent discourse*. We identify breaks in the discourse in order to find comment threads that are interleaved and hence hinder on readability and understandability.

In this work we analyze the text extracted from the titles, the descriptions and the comments in bug reports. Bug descriptions sometimes include stack traces, source code or patches, but we also treat them in this case as unstructured text (i.e., sets of words).

Many open-source bug tracking repositories, such as Bugzilla provide an interface to query and extract bug report descriptions. We developed a tool that crawls through the web interface of Bugzilla for a given project and extracts all bug reports in XML format.

3.1. Measuring coherence with Latent Semantic Analysis

In order to analyze the text from the bug reports we use Latent Semantic Analysis (LSA). LSA [6, 8] is a corpus-based statistical IR-based method for inducing and representing aspects of meanings of words and passages in natural language, which are reflective in their usage in large bodies of textual information.

Foltz et al. [9] showed that LSA can be applied as an automated method for measuring textual coherence of natural language texts. The primary method for using LSA to measure textual coherence is to compare some unit of text to an adjacent element of text in order to determine the degree to which the two are semantically related. These elements of text may be sentences, paragraphs, individual words, or even whole chapters in books.

In our case, these elements of text are unique comments in the bug reports. This analysis can then be performed for all pairs of adjacent comments in order to describe the overall coherence of the discussion.

[4] http://www.mozilla.org/bugs/

To measure the textual coherence, LSA is used to compute semantic similarities between successive comments in the bug report. High similarity between two consecutive comments means that the two comments are related, whereas low similarity indicates a break in the topic or discourse. A well written essay or paper may indicate textual coherence even at these break points, thus topic changes are not always identified by a lack of coherence. For instance, a writer may deliberately make a series of disjointed points, which may not represent a split in the discourse structure. The idea is that if the semantic similarity between neighboring sentences is maintained at a high level, the reader can follow the logic and comprehend the information in the text more easily. As the semantic similarity measure, as defined by LSA, is not transitive, it is possible to have non-adjoining sentences having low similarity measure, yet maintaining high coherence of a text. The overall coherence of a text is measured as the average of all semantic similarity measures between all the consecutive sentences (comments).

The measuring methodology for the proposed textual coherence metric is:

- Bug reports are extracted from the repository and each comment forms a document in the corpus.
- Simple tokenization techniques (e.g., identifier splitting, operator and punctuation removal) are applied on the corpus and common stop words are removed.
- LSA is used to index the corpus and create a corresponding semantic space. In this semantic space, each document from the corpus is represented by real-valued vector.
- Semantic similarities are computed between each contiguous pair of documents (comments) in the corpus.

We consider a bug-tracking system for a specific project with a set of bug reports B = $\{b_1, b_2 \ldots b_n\}$.

Each bug report is defined as b $\in$ B, b = $(d_{short}, d_{long}, \{c_1, \ldots, c_k\})$, where d_{short} is the short description of the bug (i.e., title), d_{long} is the long description and $\{c_1, \ldots, c_k\}$ is a set of comments associated with the bug report.

Definition 1. For every bug report $b_i \in$ B containing at least two comments, we define the *semantic similarity between two adjoining comments*, c_i and c_{i+1}, SSC(c_i, c_{i+1}) as the cosine between the vectors corresponding to c_i and c_{i+1} in the semantic space constructed by LSA, SSC(c_i, c_{i+1}) $\in$ [-1, 1]:

$$SSC(c_i, c_{i+1}) = \frac{vc_i^T vc_{i+1}}{|vc_i|_2 \times |vc_{i+1}|_2},$$

where vc_i and vc_{i+1} are the vectors corresponding to c_i and c_{i+1}, respectively; T denotes the transpose operation, and $|vc_i|_2$ is the length of the vector.

For each bug reports, $b_i \in$ B we compute k-1 distinct semantic similarities between adjoining comments (e.g., c_1 and c_2, c_2 and c_3, …, c_{k-1} and c_k).

Given this representation of bug reports, we define a measure that approximates the textual coherence of comments in a bug report by measuring the degree to which the comments in bug reports relate to each other.

Definition 2. *The average semantic similarity of the comments in a bug report* $b_i \in$ B is:

$$ASSC(b_i) = \frac{1}{k-1} \times \sum_{i=1}^{k-1} SSC(c_i, c_{i+1}),$$

where (c_i, c_{i+1}) are adjacent comments in the bug report, and k is the total number of comments in the bug report.

In our view, $ASSC(b_i)$ defines the degree to which comments in a bug report relate to each other.

Definition 3. *The textual coherence of a bug report* $b_i \in$ B is defined as following:

$$TCBR(b_i) = \begin{cases} ASSC(b_i) & if \quad ASSC(b_i) > 0 \\ else & 0 \end{cases}$$

Based on the above definitions, $TCBR(b) \in [0, 1] \ \forall$ b $\in$ B. If comments in a bug report $b_i \in$ B are textually coherent, then the value of $TCBR(b_i)$ should be closer to one, meaning that adjacent comments in a bug report relate textually to each other (i.e., the SSC for each adjacent pair of comments is close to one, meaning that they used similar words). In this case, all the comments in a bug report discuss the same problem without major changes in the topic or breaks in discourse structure. If the comments inside the bug report have low semantic similarity values between them (i.e., the SSC for majority of pairs of comments will be close to or less than zero), then the comments most likely address different issues (e.g., separate discussion threads) and $TCBR(b_i)$ will be close to zero.

4. The Coherence of Bug Reports in Eclipse

In order to evaluate our novel measure for capturing textual coherence of comments in bug reports, an experienced developer (subject) rated how well the automatically computed values of *TCBR* match actual coherence of comments in bug reports. The subject, who participated in this evaluation, is a graduate student majoring in Computer Science with five years of programming experience in Eclipse and one year experience working with Bugzilla.

In this section we present the details on how bug reports are mined, selected and indexed for evaluation.

4.1. Object of the study

We extracted a number of bug reports for Eclipse 2.0 software project, which is hosted on Bugzilla. Originally, we extracted 3,401 bug reports, which have been officially resolved (or fixed). Preliminary lexical analysis of these bug reports generated the following statistical facts. The average number of comments per bug report is 4.9; standard deviation is 4.5 (with distribution of comments in bug reports shown in Figure 1). The average number of words in the comments is 22.5. The total number of unique words in comments is 15,748 (not including unique words in descriptions). For the study we decided to keep bug reports which contain at least *four* comments. After filtering out the bug reports containing less than four comments, we kept 1,763 bug reports.

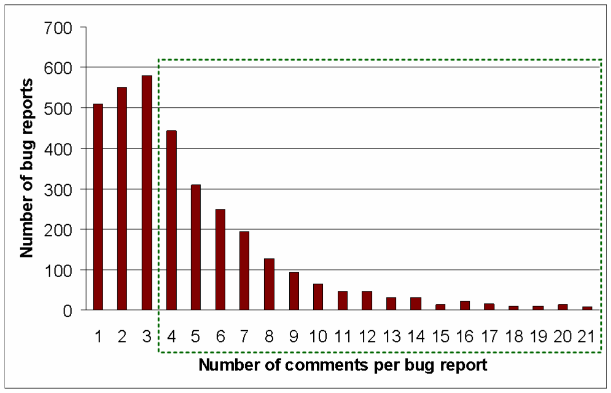

Figure 1. The number of comments in all 3,401 bug reports in Eclipse 2.0. In the study we used 1,763 bugs which contained at least 4 comments.

To build the corpus, we extracted all the comments from the 1,763 bug reports and the long and short descriptions. Each bug report description and each comment from the bug report was represented as a separate document in the corpus. We applied tokenization in all the documents in corpus, as described above and removed common stop words. Where identifiers were split, the original form was also kept. For example, “BugReport” was split into “bug” and “report” and “BugReport” was kept in the corpus. The Porter stemmer [12] was also used to stem the words in the corpus. The resulting corpus consisted of 15,087 documents (descriptions + comments) with 7,244 uniquely indexed words (with 9,526 unique words before stemming). We used a dimensionality reduction factor of 300 for LSA indexing.

4.2. Evaluation setup

In order to gain more insights into the *TCBR* measure, and how it reflects textual coherence of comments in bug reports, the subject was given a set of ten bug reports for which we computed the *TCBR* measures. For these ten bug reports, we randomly selected five bug reports from the 10% of bug reports with highest values of *TCBR* and five bug reports from

Table 1. Examples of ten bug reports with low and high textual coherence of comments, with the developer's assessment. The developer's assessments include suggestions on which comments could be grouped into multiple discussion threads (numbers in parenthesis indicate the comment number).

Bug ID	No. of comments	Average similarity	Minimum similarity	Maximum similarity	Developer's assessment
		Low			
13512	4	0	0	0	*Poor* : 3 threads: (1) (2, 4) (3)
13584	6	0	0	0.0024	*Fair*: 2 threads: (1, 3, 4, 6) (2, 5)
14646	7	0.0332	0.0034	0.1205	*Poor*: 3 threads: (1, 5, 7) (2, 3, 4) (6)
14326	4	0	0	0.0003	*Fair*: 2 threads: (1, 2, 3) (4)
26127	6	0.0004	0	0.002	*Fair*: 2 threads: (1, 3) (2, 4, 5, 6)
		High			
25990	5	0.339	0.0456	0.7057	*Very High*: 1 thread: (1, 2, 3, 4, 5)
15687	5	0.3454	0.1302	0.5347	*Very High*: 1 thread: (1, 2, 3, 4, 5)
13656	6	0.3573	0.1852	0.6242	*Very High*: 1 thread: (1, 2, 3, 4, 5, 6)
8969	5	0.3821	0.2085	0.6042	*High*: 1 thread: (1, 2, 3, 4, 5)
12952	5	0.4012	0.0234	0.7337	*High*: 1 thread: (1, 2, 3, 4, 5)

the 10% of bug reports with lowest values of *TCBR*. These ten bug reports were shuffled and presented to the developer without the actual values of *TCBR* measure. The subject was asked to read each of the bug report with its comments and classify them based on the following scale: *(1) Very high* coherence of comments – the comments address the problem coherently with a single discussion thread with no breaks in discourse; *(2) High* coherence of comments – the comments address the problem coherently with minor breaks in discourse (some other minor issues might be discussed) with a single discussion thread; *(3) Fair* coherence of comments – the comments refer to more than one problem, however no more than two discussion threads are identified; *(4) Poor* coherence of comments – the comments refer to more than two separate problems and, therefore, can be structured in more than two discussion threads.

4.3. Discussion of the results

The results of assessing the sampled bug reports are presented in Table 1.

The subject identified that bug #13512[5] has *poor* coherence. He pinpointed three separate discussion threads: the first one, which consists of comment #1 and provides clarification to the initial description; the second thread, which consists of comments #2 and #4 (comment #4 is a reply to comment #2); and the third thread consists of comment #3, which specifies additional information about duplicate bugs.

For bug #13584[6] (rated *fair*), the subject identified two discussion threads: the first one deals with the solution to the problem described in the bug report (comments #1, #3, #4, #6) and the second one deals with related bugs (comment #2) and duplicates (comment #5).

The subject rated the bug #14646[7] as having a *poor* coherence. Comments #1, #2, and #6 propose three different solutions for the problem, which could be categorized in three different discussion threads. Comments #3 and #4 follow up on comment #2, creating the thread between comments #2, #3, and #4, whereas comments #5 and #7 are responses to comment #1, thus creating the thread between comments #1, #5, and #7. On the other hand, comment #7 also addresses the issue raised in comment #6, which indicates that some comments are responses to multiple discussion threads. Grouping these comments into three different discussion threads would improve comprehensibility of this bug report.

The subject identified two threads in bug report #14326[8] and thus rated it as having a *fair* coherence between comments. The first one relates to the solution (comments #1, #2, and #3) and the second thread references to a duplicate bug (comment #4).

Similarly, the subject identified two threads for bug #26127[9] (rated *fair*): the first one relates to the solution (comments #2, #4, #5, and #6) and the second one deals with duplicates of this bug (comments #1 and #3).

The subject evaluated the bug #25990[10] to have *very high* textual coherence and hence a single, coherent thread linked to the provided description, with no gaps or breaks in the discourse structure of comments.

The subject marked the bugs #15687[11] and #13656[12] each having a single, highly coherent discussion thread,

[5] https://bugs.eclipse.org/bugs/show_bug.cgi?id=13512
[6] https://bugs.eclipse.org/bugs/show_bug.cgi?id=13584

[7] https://bugs.eclipse.org/bugs/show_bug.cgi?id=14646
[8] https://bugs.eclipse.org/bugs/show_bug.cgi?id=14326
[9] https://bugs.eclipse.org/bugs/show_bug.cgi?id=26127
[10] https://bugs.eclipse.org/bugs/show_bug.cgi?id=25990
[11] https://bugs.eclipse.org/bugs/show_bug.cgi?id=15687
[12] https://bugs.eclipse.org/bugs/show_bug.cgi?id=13656

which is related to the bug description. Both of these bug reports received the highest mark – *very high*.

The subject rated the bug #8969[13] as having a *high* coherence. All the comments offer discussion about the problem, but there are some minor breaks in the flow of discourse. For example, comment #2 relates to comment #1, while comment #3 relates to comment #1 and #2 at the same time (instead of being a response to comment #2). This can be explained by the fact that the author of the comment #3 addressed a mistake made in the comment #1 and also addressed the comment #2 at the same time (in his comment #3). Similarly, the comment #4 relates more to comment #1 rather than to comment #3 or even comment #2.

The subject found a minor inconsistency in the coherence of comments for bug #12952[14] (rated as *high*). He found a clear thread consisting of comments #1, #2, #3, and #5 and established that comment #4, although related to the solution discussed in the previous comments, contains some "raw" information that is not blended in this particular discourse structure.

4.4. Limitations

Interpretation and generalization of the results has to be done with caution. We had only one subject evaluating a set of provided bug reports and comments, thus no statistical significance of the results can be established. The study used bug reports from only one software system. The size of the sample, provided for the investigation is relatively small and, thus it can serve for illustration purposes only. Also, some of the comments in the bug reports contained structural information, such as stack traces and source code (e.g., bug #25990), which could have impacted the values of the *TCBR* measure. We did not apply any transformations for misspelled words appearing in comments (e.g., bug#10468: "*deceide*", bug# 21469: "*recieve*", bug# 7557: "*acutally*") and therefore those cases could have impacted similarity values for some pairs of comments as well.

5. Conclusions and Future Work

The paper presents a novel approach to automatically measure the textual coherence of bug reports based on the analysis of the text found in the descriptions and the comments of bug reports. Our study on a subset of Eclipse bugs reports indicates that the measure is a good indicator of textual coherence of comments in bug reports, which is also confirmed by the developer.

[13] https://bugs.eclipse.org/bugs/show_bug.cgi?id=8969
[14] https://bugs.eclipse.org/bugs/show_bug.cgi?id=12952

We are planning on extracting structural information, such as stack traces and source code, from comments automatically and using that for augmenting the textual coherence measure. We are also planning on conducting several case studies using bug reports from different software systems and different bug-tracking systems. Our implementation of the measure is specific to the bug report format supported in Bugzilla, thus the measure may have to be adjusted to other formats if necessary. Moreover, we will secure several subjects for conducting evaluation to warrant statistically significant results. In addition to sampling bug reports with high and low similarity values, we will also select bug reports with other values of the metric. We will investigate an impact of automatically applying spell-checking of words in comments based on editing distance measure.

Our novel measure lays the foundation for several possible applications. One of the first applications relates to automatic assessment of bug report quality based on the textual information in descriptions and comments. While in this work, we defined the measure which captures semantic similarities among comments only, in the future we will define measures taking into account similarities among bug descriptions and comments. We will also work on approaches which can use our measure for automatic categorization of comments into discussion threads.

6. Acknowledgements

This research was supported in part by a grant from the US National Science Foundation (CCF-0438970). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not reflect the views of the NSF.

7. References

[1] Anvik, J., Hiew, L., and Murphy, G. C., "Who should fix this bug?" in Proc. of 28th International Conference on Software Engineering (ICSE'06), 2006, pp. 361-370.

[2] Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., and Zimmermann, T., "Quality of Bug Reports in Eclipse", in Proc. of 2007 OOPSLA Workshop on Eclipse Technology EXchange, 2007, pp. 21-25.

[3] Canfora, G. and Cerulo, L., "Impact Analysis by Mining Software and Change Request Repositories", in Proc. of 11th IEEE International Symposium on Software Metrics (METRICS'05), September 19-22 2005, pp. 20-29.

[4] Canfora, G. and Cerulo, L., "Fine Grained Indexing of Software Repositories to Support Impact Analysis", in Proc. of International Workshop on Mining Software Repositories (MSR'06), 2006, pp. 105 - 111.

[5] Cubranic, D. and Murphy, G. C., "Automatic Bug Triage Using Text Categorization", in Proc. of 6th International

Conference on Software Engineering & Knowledge Engineering (SEKE'04), 2004, pp. 92–97.

[6] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R., "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, vol. 41, 1990, pp. 391-407.

[7] Di Lucca, G. A., Di Penta, M., and Gradara, S., "An Approach to Classify Software Maintenance Requests", in Proc. of IEEE International Conference on Software Maintenance (ICSM'02), Montréal, Québec, Canada, 2002, pp. 93-102.

[8] Dumais, S. T., "Improving the retrieval of information from external sources", *Behavior Research Methods, Instruments, and Computers*, vol. 23, no. 2, 1991, pp. 229 - 236.

[9] Foltz, P. W., Kintsch, W., and Landauer, T. K., "The Measurement of Textual Coherence with Latent Semantic Analysis", *Discourse Processes*, vol. 25, no. 2, 1998, pp. 285-307.

[10] Hooimeijer, P. and Weimer, W., "Modeling Bug Report Quality", in Proc. of 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), November 5-9 2007, pp. 34-43.

[11] Ko, A. J., Myers, B. A., and Chau, D. H., "A Linguistic Analysis of How People Describe Software Problems in Bug Reports", in Proc. of IEEE Conference on Visual Language and Human-Centric Computing (VL/HCC), 2006, pp. 127-134.

[12] Porter, M., "An Algorithm for Suffix Stripping", *Program*, vol. 14, no. 3, July 1980, pp. 130-137.

[13] Runeson, P., Alexandersson, M., and Nyholm, O., "Detection of Duplicate Defect Reports Using Natural Language Processing", in Proc. of 29th IEEE/ACM International Conference on Software Engineering (ICSE'07), Minneapolis, MN, 2007, pp. 499-510.

[14] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J., "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information", in Proc. of 30th International Conference on Software Engineering (ICSE'08), Leipzig, Germany, 10 - 18 May 2008 2008.

[15] Weimer, M., Gurevych, I., and Muhlhauser, M., "Automatically Assessing the Post Quality in Online Discussions on Software", in Proc. of Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 2007, pp. 125-128.

[16] Weiß, C., Premraj, R., Zimmermann, T., and Zeller, A., "How Long Will It Take to Fix This Bug?" in Proc. of 4th Working Conference on Mining Software Repositories (MSR'07), 2007.