Volume 3, Issue 11, November-2016, pp. 585-588 ISSN (O): 2349-7084

International Journal of Computer Engineering In Research Trends

Available online at: www.ijcert.org

Software Engineering Domain Knowledge to Identify Duplicate Bug Reports

*Dr.J.KEZIYA RANI

ASST.PROFESSOR, DEPT.OF.CS&T,
S.K.UNIVERSITY, ANANTHAPURAMU.

Abstract- Earlier, many methodologies was proposed for detecting duplicate bug reports by comparing the textual content of bug reports to subject-specific contextual material, namely lists of software-engineering terms, such as non-functional requirements and architecture keywords. When a bug report includes a word in these word-list contexts, the bug report is measured to be linked with that context and this information is likely to improve bug-deduplication methods. Here, we recommend a technique to partially automate the extraction of contextual word lists from software-engineering literature. Evaluating this software-literature context technique on real-world bug reports creates useful consequences that indicate this semi-automated method has the potential to significantly decrease the manual attempt used in contextual bug deduplication while suffering only a minor loss in accuracy.

Key words— software literature; duplicate bug reports; information retrieval; machine learning; documentation.

I. INTRODUCTION

An essential component of the quality-assurance processes for many modern software projects is the use of issue-tracking systems; these systems trace the bug reports or, testers, issues that developers and users meet for a particular software system. As an outcome, these systems supplies as repositories of bug reports, stack traces, and feature requests. Issue tracker represents a proxy for measuring the developers' productivity based on their development in correcting or addressing an issue or bug report. Bug reports are typically written in natural-language text, which implies that the same matter can potentially be illustrated in many ways by the various users or developers that encounter this issue. Normally the vocabulary used by developers differs from that used by users; therefore, to recognize that two bug reports refer to the same bug, a triage, often an experienced developer, has to use their expert knowledge to convert bug reports into a technical language that developers can understand. Often these reports are duplicates of existing bug-reports that have been or are currently being addressed. Identifying duplicate reports is a significant problem that, if solved, would allow developers to fix bugs faster, and prevent them from wasting time by addressing the same bug several times.

In order to decrease manual triaging attempt, substantial research has been made to find an automated method of detecting duplicate bugs. Prior work from Runeson et al. [1] and Sun et al. [2], [3] in bug report de-duplication, detection of duplicate bug reports, measures bug report similarity by joining natural language similarity measures of bug report descriptions with categorical bug attributes such as "component", "type", and "priority". Normally these methods use off-the-shelf document-similarity measures and relate them to bug reports. While this is efficient, it ignores an important context.

Keeping this in mind as important fact, Alipour et al. [4], [5] exploited the software-engineering and project-specific context to boost bug-report deduplication. By making use of contextual data and comparing a bug report to terms referring to non-functional requirements or architectural descriptions, Alipour et al. enhanced the bug-deduplication performance of Sun et al. [3]. This contextual method tested manually created word lists and topics generated through organized labelled Latent Dirichlet Allocation (labelled-LDA) on the project's bug descriptions. Labelled-LDA worked well but is an effort-intensive process [6]. Alipour gave a conclusion that contextual features tend to expose

 DOI:10.22362/ijcert/2016/v3/i11/XXXX

the relationship between the bug report text and concepts such as non-functional requirements or architectural modules. These relationships can be exploited in bug report deduplication when different terminology is used to describe the same situation. Therefore not only could one compare text between bug reports, one could compare their contexts and associations as well.

The technique presented in this paper, called the software-literature context method, make use of context and further decreases the manual effort linked with detecting duplicate bug reports with generic contextual features extracted from software-engineering literature, rather than project-specific analysis. These features are sufficient to apply to any software-development project. The proposed method is different from the method described by Alipour et. al. as it uses word lists extracted from software literature instead of using bug reports from the software projects to extract the word lists. These software-literature context word lists reflect the software-development processes, and the evolution of project, and hence show the bug report descriptions.

II.RELATED WORK

The majority bug-deduplication methods use textual analysis to detect duplicate bug reports. Runeson et al. [1] used NLP techniques to detect 66% of the duplicate reports of Sony Ericson Mobile Communications. Bettenberg et al. [10] used machine-learning classifiers to triage the reports based on representing the report titles and descriptions as word vectors. Using support vector machines (SVM) and naïve Bayes, they obtained accuracy scores of roughly 65%.

Wangetal. [12] projected a method using implementation traces on the Eclipse dataset using NLP techniques; on the other hand, their methodology relies on the manual extraction of the execution traces, which makes it extremely time-intensive.

Surekha et al. [13] used an n-gram based textual model on an Eclipse dataset to report top-k bug reports that could be duplicate of a given bug report. Building on their work, Sun et al. [3] projected a new model based on the BM25F score that uses the term frequency-inverse document frequency (TF-IDF) score to measure report similarity. Along with this, they use definite features such as priority and severity to create substantial development over previous methods. Sun et al. sorted the reports into different "buckets" according to the underlying bugs, and paying attention on sending incoming duplicate reports to the suitable bucket. They report top-k bug reports for a given bug that could be duplicate and use the bugs marked as duplicates of the bug by triager from the original dataset to find an improvement of 10-27% over Surekha et al.. Only the concern with this kind of evaluation is only true-positives are queried. Bug reports with no duplicates are not evaluated or queried, thus true-negatives are not tested.

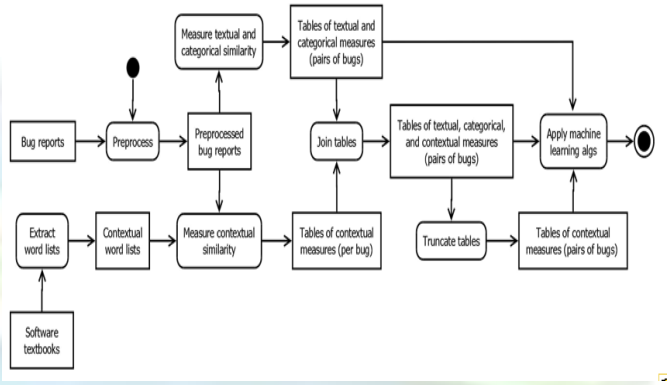

Fig.1.Workflow of the software-literature context evaluation methodology showing inputs and output.

The sharp edged rectangles represent data and the rounded corner rectangles represent activities.

Jalbert et al. [11] used categorical features of bug reports, textual-similarity and graph-clustering methods to filter out duplicate reports. They developed an automatic method to detect the duplicate bug reports, and allowed only one of the duplicates to reach developers. They calculated their technique on a dataset of 29; 000 bug reports from Mozilla Firefox and were able to filter out 8% of the duplicate bug reports.

Alipour [4], [5] enhanced the work of Sun et al. [3] by adding contextual features using both labelled and unlabelled LDA generated word lists [6] to the method used by Sun et al.. Alipour et al. reformulated the task as detecting whether given pair of bugs are duplicates or not. The use of LDA formed strong improvements in accuracy, increasing by 16% over the results obtained by Sun et al.. Klein et al. [9] and Lazar et. al. [14] have leveraged the same dataset against new textual metrics based LDA's output to achieve an correctness improvement of 3% over Alipour et al.. The work is capable but relies on running LDA on the corpus itself whereas some of the features illustrated in this paper are digged out only once from textbooks and can be applied broadly to other software projects without any extraction effort from the client.

III.METHODOLOGY

In this section we describe how to measure the efficiency of using contextual features, such as similarity to software engineering texts, to decide if bug reports are duplicates of each other. First the processes of creation of the datasets and the contextual word-lists used to measure contextual features are described. Secondly, the extraction of contextual features is discussed. Finally, the assessment of how well these contextual features help bug-report deduplication is discussed.

A. Contextual-Features Extraction:

The following word lists were used as context in this study:

1) General software engineering: This word list was extracted from Pressman's textbook [7]. The book was split into 13 different word lists corresponding to its chapters like architecture, UI design, formal methods, and testing. The complete process took around half an hour for a single person.

2) Eclipse documentation: This word list was extracted from the Eclipse platform documentation for Eclipse 3:1 [15]. The documentation was split into 19 different word lists relating to using Eclipse, debugging, and IDE features. The process around half an hour for a single person.

3) Mozilla documentation: This word list was extracted from the online developer guide for Mozilla [17]. Unlike Eclipse, and Open Office there is no one central documentation for Mozilla products; the online documentation consists of several webpages, with very short descriptions catering to online audience. The documentation was split into 13 different word lists relating to using various components such as browser JavaScript, debugging tools, and testing. The process around half an hour for a single person.

4) Android development: This word list was extracted from the chapters of Murphy [8] to produce ten word lists describing features like widgets, activities, databases. They were all related specifically to Android application Development, and the process took around half an hour for a single person.

5) Random English Words: In order to determine that the specialized word lists were actually having a significant effect, random English word lists were used as well. These word lists are the same as those used by Alipour et al., consisting of 26 lists each with 100 random words.

6) LDA: These word lists were extracted using topic modelling on all the bug reports from the Eclipse, Mozilla, and Open Office bug dataset by Alipour et al. 20 word lists were constructed by using LDA for all the three projects, and the process took around one person-hour for each project. The topics were unlabelled.

7) Labelled LDA: These word lists were extracted using labelled LDA on the Android bug reports by Dan et al. [6]. There are 72 word lists on a variety of subjects like wifi, GPS, 3G, keyboard, that were extracted with 60 person-hours of effort on Android bug data-set [18]. These word lists are not available for other 3 projects – Eclipse, Mozilla, and Open Office.

The first four lists, general software engineering, Android development, Eclipse documentation, and Mozilla documentation, were extracted by labelling chapters on the basis of the software-engineering processes like maintenance, testing. The last three lists were used by Alipour et al.. These word lists contain words that are more inclined to describe the software engineering processes while the ones used by Alipour et al. depict the products, and specific problems in the software development in the bug reports.

B. Bug-Report Preprocessing:

The bug reports were pre-processed according to the methodology of Alipour et al.. Bugs missing enough information to use in the deduplication process were useless. This included bugs without Bug IDs as well as bugs marked as a duplicate where the corresponding duplicate Bug ID was not found in the repository. Stop words were detached from the description and title fields using a comprehensive list of English stop words.

The reports were organized into "buckets", similar to the methodology of Sun et al. [3]. Each bucket includes a master bug report, along with all duplicates of that report. The master bug report is the report with the initial open time in that bucket.

A bucket that enclosed a very large set of duplicate bugs was detached from the Android dataset. This bucket would have introduced a strong bias in the results since such large clusters of duplicate bugs are unusual in other bug repositories. After this pre-processing step, three different subsets of the bug reports were built for the Android, Eclipse, Mozilla, datasets, each includes a different ratio of duplicate to non-duplicate reports, in order to watch the effect of different ratios. The contradictory ratios were used to decide what effect the proportion of duplicate bug

reports had on the results. The three subsets used included a set with 20% duplicates (as per Alipour et al.), 10% duplicates, and 30% duplicates. In each case, random selection without substitute from the original dataset was used, selecting as many reports as possible while maintaining the desired ratios. The focus, however, was on the 20% duplicate to 80% non-duplicate split used by Alipour et al.; A comparison between results using software-literature context features with those achieved using LDA is also presented, along with results achieved using entirely random word lists, as a common sense check.

IV.CONCLUSION

This work shows a method of improving the detection of duplicate bug reports using contextual information extracted from software-engineering textbooks and project documentation. This paper proposes a simpler software-literature context method for timely detection of duplicate bug reports, supports the previous findings of Alipour et al. that include contextual data into software-engineering classification systems can lead to decrease in the manual effort essential while maintaining or improving the accuracy of duplicate bug report detection systems.

REFERENCES

[1] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in Software Engineer-ing, 2007. ICSE 2007. 29th International Conference on. IEEE, 2007, pp. 499–510.

[2] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in Pro-ceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, 2010, pp. 45–54.

[3] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011, pp. 253–262.

[4] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in Proceedings of the Tenth International Workshop on Mining Software Repositories. IEEE Press, 2013, pp. 183–192.

[5] A. Alipour, "A contextual approach towards more accurate duplicate bug report detection," Master's thesis, University of Alberta, Fall 2013.

[6] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding android fragmentation with topic analysis of vendor-specific bugs," in Reverse Engineering (WCRE), 2012 19th Working Conference on. IEEE, 2012, pp. 83–92.

[7] R. S. Pressman and W. S. Jawadekar, "Software engineering," New York 1992, 1987.

[8] M. L. Murphy, The Busy Coder's Guide to Advanced Android Develop-ment. CommonsWare, LLC, 2009.

[9] N. Klein, C. S. Corley, and N. A. Kraft, "New features for duplicate bug detection." in MSR, 2014, pp. 324–327.

[10] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful. . . really?" in Software Maintenance, 2008. ICSM 2008. IEEE International Conference on. IEEE, 2008, pp. 337–345.

[11] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on. IEEE, 2008, pp. 52–61.

[12] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel, "Rebucket: A method for clustering duplicate crash reports based on call stack similarity," in Proceedings of the 2012 International Conference on Software Engineering. IEEE Press, 2012, pp. 1084–1093.

[13] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in Software Engineering Conference (APSEC), 2010 17th Asia Pacific. IEEE, 2010, pp. 366–374.

[14] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014, pp. 308–311.

[15] A. Kiezun. Basic tutorial eclipse 3.1. [On-line]. Available: http://archive.eclipse.org/eclipse/downloads/drops/R-3. 1-200506271435/org.eclipse.jdt.doc.user.3.1.pdf.zip

[16] Sun Microsystems. (2008) Openoffice.org 3.0 developer's guide. [Online]. Available: https://wiki.openoffice.org/w/images/3/34/Developers Guide_ OOo3.0.0.odt

[17] Mozilla Developer Network and individual contributors. Mozilla developer guide. [Online]. Available: https://developer.mozilla.org/enUS/docs/Mozilla/Developer_guide

