Volume 2, Issue 6, June 2017 ISSN (online): 2456-0006
International Journal of Science Technology Management and Research
Available online at: www.ijstmr.com

Automatic Summarization of Bug Reports and Bug Triage classification

Prajakta Kokate[1]
Department of Computer Engineering
L.G.N.Sapkal COE,Anjaneri,Nashik,
MH, India

N.R.Wankhade[2]
Department of Computer Engineering
L.G.N.Sapkal COE,Anjaneri,Nashik
MH, India

Abstract: To help with a number of different tasks, software developers access bug reports in a projects bug repository, including understanding how previous changes have been made and understanding multiple aspects of particular defects. A substantial amount of text is often required by a developer interaction with existing bug reports. The developers can perform their tasks by consulting shorter summaries instead of entire bug reports, in this article we investigate whether it is possible to summarize bug reports automatically. The quality of generated summaries is similar to summaries produced for email threads and other conversations, we investigated whether existing conversation based automated summarizers are applicable to bug reports. A summarizer on a bug report corpus is trained by us. By existing conversation based generators, this summarizer produces summaries that are statistically better than summaries produced. We conducted a task based evaluation that considered the use of summaries for bug report duplicate detection tasks, to determine if automatically produced bug report summaries can help a developer with their work. We found that summaries helped the study participants save time, that there was no evidence that accuracy degraded when summaries were used and that most participants preferred working with summaries to working with original bug reports.

Keywords-: **Empirical software engineering, summarization of software artifacts, bug report duplicates detection, Bug Classification.**

I. INTRODUCTION

A rich source of information for a software developer working on the project, a software project bug repository provides. To understand how changes were made on the project in the past, For instance, the developer may consult the repository to understand reported defects in more details. Either as the result of a search or a recommendation engine, when accessing the projects bug repository, a developer often ends up looking through a number of bug reports. A developer must peruse are relevant to the task at hand, typically only a few of the bug reports. Other times a developer must read the report, sometimes a developer can determine relevance based on a quick read of the title of the bug report, which can be lengthy, involving discussions amongst multiple team members and other stakeholders. For example, a developer using the bug report duplicate recommender built by Sun et al. [2] to get a list of potential duplicates for bug #564243 from the Mozilla system,1 is presented with a total of 5,125 words (237 sentences) in the top six bug reports on the recommendation list.

Allow a developer to more efficiently investigation information in a bug repository as part of a task, in this paper, we investigate whether concise summaries of bug reports, automatically produced from a complete bug report. Its authors would write a concise summary that represents information in the report to help other developers who later access the

report, perhaps optimally, when a bug report is closed. Given the evolving nature of bug repositories and the limited time available to developers, this optimal path is unlikely to occur. As a result, we investigate the automatic production of summaries to enable generation of up-to-date summaries on-demand and at a low cost. Bug reports vary in length. Consisting of only a few words some are short. Between many developers and user, others are lengthy and include conversations. Displays part of a bug report from the KDE bug repository;2 the entire report consists of 21 comments from six people.

Unlikely to benefit from summaries of short bug reports, developers may benefit from summaries of lengthy bug reports. A common size requirement for short paragraph length summaries, if we target summaries of 100 words, and we assume a compression rate of 33 percent or less is likely beneficial, then a bug report must be at least 300 words in length to be worth summarization. Over the two year period of 2011 2012, the number and percentage of summary worthy bug reports in three popular and large scale open source software projects. Suggesting that sufficient lengthy bug reports exist to make the automatic production of summaries worthwhile, in all these projects, almost one third of bug reports created over the 24 month period are longer than 300 words. The length of a bug report is the total number of words in its description and comments.

II. RELATED WORK

Analyzing candidate traceability links is a difficult, time consuming and error prone task, as it usually requires a detailed study of a long list of software artifacts of various kinds. One option to alleviate this problem is to select the most important features of the software artifacts that the developers would investigate. We discuss in this position paper how text summarization techniques could be used to address this problem. The potential gains in using summaries are both in terms of time and correctness of the traceability link recovery process.

In a bug tracking system, different testers or users may submit multiple reports on the same bugs, referred to as duplicates, which may cost extra maintenance efforts in triaging and fixing bugs. In order to identify such duplicates accurately, in this paper we propose a retrieval function (REP) to measure the similarity between two bug reports. It fully utilizes the information available in a bug report including not only the similarity of textual content in *summary* and *description* fields, but also similarity of non-textual fields such as *product*, *component*, *version*, etc. For more accurate measurement of textual similarity, we extend *BM25F* – an effective similarity formula in information retrieval community, specially for duplicate report retrieval. Lastly we use a two-round stochastic gradient descent to automatically optimize REP for specific bug repositories in a supervised learning manner. We have validated our technique on three large software bug repositories from Mozilla, Eclipse and Open Office. The experiments show 10–27% relative improvement in *recall rate@k* and 17–23% relative improvement in *mean average precision* over our previous model. We also applied our technique to a very large dataset consisting of 209,058 reports from Eclipse, resulting in a *recall rate@k* of 37–71% and *mean average precision* of 47%. Software maintenance is a relevant and expensive phase of the software development process. Developers have to deal with legacy and undocumented code that hinders the comprehension of the software system at hand. Enhancing program comprehension by means of recommender systems in the Integrated Development Environment (IDE) is a solution to assist developers in these tasks. The recommender systems proposed so far generally share common

Weaknesses: they are not proactive, they consider a single type of data-source, and in case of multiple data-sources, and relevant items are suggested together without considering interactions among them. We envision a future where recommender systems follow a holistic

Approach: They provide knowledge regarding a programming context by considering information beyond the one provided by single elements in the context of the software development. The recommender system should consider

different elements such as development artifact (e.g., bug reports, mailing lists), and online resources (e.g., blogs, Q&A web sites, API documentation), developers activities, repository history etc. The provided information should be novel and emerge from the semantic links created by the analysis of the interactions among these elements.

Most speech summarization research is conducted on broadcast news. In our viewpoint, spontaneous conversations are a more "typical" speech source that distinguishes speech summarization from text summarization, and hence a more appropriate domain for studying speech summarization. For example, spontaneous conversations contain more spoken-language characteristics, e.g. disfluencies and false starts. They are also more vulnerable to ASR errors. Previous research has studied some aspects of this type of data, but this paper addresses the problem further in several important respects. First, we summarize spontaneous conversations with features of a wide variety that have not been explored before. Second, we examine the role of disfluencies in summarization, which in all previous work was either not explicitly handled or removed as noise. Third, we breakdown and analyze the impact of WER on the individual features for summarization.

III. LITERATURE SURVEY

For many software projects, bug tracking systems play a central role in supporting collaboration between the developers and the users of the software. To better understand this collaboration and how tool support can be improved, we have quantitatively and qualitatively analyzed the questions asked in a sample of 600 bug reports from the MOZILLA and ECLIPSE projects. We categorized the questions and analyzed response rates and times by category and project. Our results show that the role of users goes beyond simply reporting bugs: their active and ongoing participation is important for making progress on the bugs they report. Based on the results, we suggest four ways in which bug tracking systems can be improved.

Publicly accessible bug report repositories maintained by free / open source development communities provide vast stores of data about distributed software problem management (SWPM). Qualitative analysis of individual bug reports, texts that record community responses to reported software problems, shows how this distributed community uses its SWPM process to manage software quality. We focus on the role of one basic social process, *negotiation*, in SWPM. We report on the varieties and frequencies of negotiation practices and demonstrate how instances of negotiation in different contexts affect the organization of information, the allocation of community resources, and the disposition of software problems.

Issue tracking systems help organizations manage issue reporting, assignment, tracking, resolution, and archiving. Traditionally, it is the Software Engineering community that researches issue tracking systems, where software defects are reported and tracked as 'bug reports' within an archival database. Yet, as issue tracking is fundamentally a social process, it is important to understand the design and use of issue tracking systems from that perspective. Consequently, we conducted a qualitative study of issue tracking systems as used by small, collocated software development teams. We found that an issue tracker is not just a database for tracking bugs, features, and inquiries, but also a focal point for communication and coordination for many stakeholders within and beyond the software team. Customers, project managers, quality assurance personnel, and programmers all contribute to the shared knowledge and persistent communication that exists within the issue tracking system. These results were all the more striking because in spite of teams being collocated—which afforded

Frequent, face-to-face communication—the issue tracker was still used as a fundamental communication channel. We articulate various real-world practices surrounding issue trackers and offer design considerations for future systems.

Data Quality in Defect Prediction-:

In our work, we address the problem of data reduction for bug triage. To our knowledge, no existing work has investigated the bug data sets for bug triage. In a related problem, defect prediction, some work has focused on the data quality of software defects. In contrast to multiple-class classification in bug triage, defect prediction is a binary class classification problem, which aims to predict whether a software artifact (e.g., a source code file, a class, or a module) contains faults according to the extracted features of the artifact. In software engineering, defect prediction is a kind of work on software metrics. To improve the data quality, We examine the techniques on feature selection to handle imbalanced defect data. Shivaji et al. proposes a framework to examine multiple feature selection algorithms and remove noise features in classification-based defect prediction. Besides feature selection in defect prediction, We present how to measure the noise resistance in defect prediction and how to detect noise data. Moreover, Bishnu and Bhattacherjee process the defect data with quad tree based k-means clustering to assist defect prediction

IV.PROPOSED SYSTEM

It demonstrates that it is possible to generate accurate summaries for bug reports. It reports on the creation of an annotated corpus of 36 bug reports chosen from four different systems. It demonstrates that while existing classifiers trained for other conversation-based genres can work reasonably well, a classifier trained specifically for bug reports scores the highest on standard measures. It reports on a task-based evaluation that showed bug report summaries can help developers working on duplicate detection tasks save time with no evidence that the accuracy has been impaired. Our goal in this paper is an end-to-end investigation of bug report summarization; starting with investigating whether summaries can be produced automatically for bug reports and ending by investigating whether automatically generated summaries are useful for software developers. This paper extends our previously presented work on the generation of bug report summaries with a task-based study to evaluate the usefulness of bug report summaries in the context of a real software task: bug report duplicate detection. It also clarifies the description of the approach and provides a more thorough review of related work.

The proposed method introduces a bug localization technique that considers similarity between bug reports and source codes, frequency of fixed source files and occurrence of classes and method names in the bug report to effectively locate bugs. By analyzing the bug reports, it has been seen that a file which is changed frequently to fix a bug, is changed again to fix another one when new bug reports are submitted. As the frequently changed files are more likely to contain bugs, these files should be considered to compute ranking scores. The proposed system also considers class and method names that appear in the bug report because these class and method names are directly mentioned in the bug report and so, these should be given more priority when a final score is calculated.

CONCLUSION AND FUTURE WORK

In this paper, we have investigated the automatic generation of one kind of software artifact, bug reports, to provide developers with the benefits others experience daily in other domains. We found that existing conversation-based extractive summary generators can produce summaries for reports that are better than a random classifier. We also found that an extractive summary generator trained on bug reports produces the best results. We showed that generated bug report summaries could help developers perform duplicate detection tasks in less time with no indication of accuracy degradation, confirming that bug report summaries help software developers in performing software tasks.

REFERENCES

1. D. Cubrani_c and G.C. Murphy, "Hipikat: Recommending Pertinent Software Development Artifacts," Proc. 25th Int'l Conf. Software Eng. (ICSE '03), pp. 408-418, 2003.
2. C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards More Accurate Retrieval of Duplicate Bug Reports," Proc. 26th Int'l Conf. Automated Software Eng. (ASE '11), pp. 253-262, 2011.
3. P. Over, H. Dang, and D. Harman, "DUC in Context," Information Processing and Management: An Int'l J., vol. 43, no. 6, pp. 1506-1520, 2007.
4. S. Rastkar, G.C. Murphy, and G. Murray, "Summarizing Software Artifacts: A Case Study of Bug Reports," Proc. 32nd Int'l Conf. Software Eng. (ICSE '10), vol. 1, pp. 505-514, 2010.
5. A. Nenkova and K. McKeown, "Automatic Summarization," Foundations and Trends in Information Retrieval, vol. 5, no. 2/3, pp. 103-233, 2011.
6. K. Zechner, "Automatic Summarization of Open-Domain Multiparty Dialogues in Diverse Genres," Computational Linguistics, vol. 28, no. 4, pp. 447-485, 2002.
7. X. Zhu and G. Penn, "Summarization of Spontaneous Conversations," Proc. Ninth Int'l Conf. Spoken Language Processing (Interspeech '06-ICSLP), pp. 1531-1534, 2006.
8. C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng., 2011, pp. 253–262.
9. J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone based multilevel algorithm," IEEE Trans. Softw. Eng., vol. 38, no. 5, pp. 1195–1212, Sept./Oct. 2012.
10. G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with tossing graphs," in Proc. Joint Meeting 12th Eur. Softw. Eng. Conf. 17th ACM SIGSOFT Symp. Found. Softw. Eng., Aug. 2009, pp. 111–120.
11. J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng., Jul. 2010, pp. 209–214.
12. J. W. Park, M. W. Lee, J. Kim, S. W. Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting systems,"
13. A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in Proc. 7th IEEE Working Conf. Mining Softw. Repositories, May 2010, pp. 1–10. in Proc. 25th Conf. Artif. Intell., Aug. 2011, pp. 139–144.