International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Multiannotator Approach to Analyze Document Search Results from Web Databases

Ajay Kumar

Department of Computer Applications, Sri Padmavathi college of Computer science and Technology

ABSTRACT—

When a new bug report is received, developers usually need to reproduce the bug and perform code reviews to find the cause, a process that can be tedious and time consuming. A tool for ranking all the source files with respect to how likely they are to contain the cause of the bug would enable developers to narrow down their search and improve productivity. This paper introduces an adaptive ranking approach that leverages knowledge through functional decomposition of source code, API descriptions of library components, the bug fixing history, the code change history, and the file dependency graph. Given a bug report, the ranking score of each source file is computed as a weighted combination of an array of features, where the weights are trained automatic ally on previously solved bug reports using a learning to rank technique. We evaluate the ranking system on six large scale open source Java projects, using the before fix version of the project for every bug report. **Experimental** results show that the **"rank training"** approach outperforms **the last** three **modern** methods. In particular, our method **provides good** recommendations within **of** the top 10 source files for over **70%** of bug reports **on** the Eclipse **platform** and Tomcat projects.

Index Terms—Bug reports, software maintenance, learning to rank

1. INTRODUCTION

A software *bug* or *defect* is a coding mistake that may cause an unintended or unexpected behavior of the software component [12]. Upon discovering an abnormal behavior of the software project, a developer or a user will report it in a document, called a *bug report* or *issue report*. A bug report provides information that could help in fixing a bug, with the overall aim of improving the software quality. A large number of bug reports could be opened during the development life-cycle of a software product. For example, there were 3,389 bug reports created for the Eclipse Platform product in 2013 alone. In a software team, bug reports are extensively used by both managers and devolpers in their daily development process [15]. A developer who is assigned a bug report usually needs to reproduce the abnormal behavior and perform code reviews [4] in order to find the cause. However, the diversity and uneven quality of bug reports can make this process nontrivial. Essential information is often missing from a bug report [11]. Bacchelli and Bird [4] surveyed 165 man- agers and 873 programmers, and reported that finding defects requires a high level understanding of the code and familiarity with the relevant source code files. In the survey, 798 respondents answered that it takes time to review unfamiliar files. While the number of source files in a project is usually large, the number of files that contain the bug is usually very small. Therefore, we believe that an automatic approach that ranked the source files with respect to their relevance for the bug report could speed up the bug finding process by narrowing the search to a smaller number of possibly unfamiliar files.

If the bug report is construed as a query and the source code files in the software repository are viewed as a collection of documents, then the problem of finding source files that are relevant for a given bug report can be modeled as a standard task in information retrieval (IR) [41]. As such, we propose to approach it as a ranking problem, in which the source files (documents) are ranked with respect to their *relevance* to a given bug report (query). In this context, relevance is equated with the likelihood that a particular source file contains the cause of the bug described in the bug report. The ranking function is defined as a weighted combination of features, where the features draw heavily on knowledge specific to the software engineering domain in order to measure relevant relationships between the bug report and the source code file. While a bug report may share textual tokens with its relevant source files, in general there is a significant inherent mismatch between the natural language employed in the bug report and the programming language used in the code [7]. Ranking methods that are based on simple lexical matching scores have suboptimal performance, in part due to lexical mismatches between natural language statements in bug reports and technical terms in software systems. Our system contains features that bridge the corresponding *lexical gap* by using project specific API documentation to connect natural language terms in the bug report with programming language constructs in the code. Furthermore, source code files may contain a large number of methods of which only a small number may be causing the bug. Correspondingly, the source code is syntactically parsed into

2. EXISTING SYSTEM

To investigate the relationships in bug data, Sandusky et al. form a bug report network to examine the dependency among bug reports. Besides studying relationships among bug reports, Hong et al. build a developer social network to examine the collaboration among developers based on the bug data in Mozilla project. This developer social network is helpful to understand the developer community and the project evolution.

By mapping bug priorities to developers, Xuan et al. identify the developer prioritization in open source bug repositories. The developer prioritization can distinguish developers and assist tasks in software maintenance.

To investigate the quality of bug data, Zimmermann et al. design questionnaires to developers and users in three open source projects. Based on the analysis of questionnaires, they characterize what makes a good bug report and train a classifier to identify whether the quality of a bug report should be improved.

Duplicate bug reports weaken the quality of bug data by delaying the cost of handling bugs. To detect duplicate bug reports, Wang et al. design a natural language processing approach by matching the execution information.

DISADVANTAGES OF EXISTING SYSTEM

Traditional software analysis is not completely suitable for the large-scale and complex data in software repositories.

In traditional software development, new bugs are manually triaged by an expert developer, i.e., a human triager. Due to the large number of daily bugs and the lack of expertise of all the bugs, manual bug triage is expensive in time cost and low in accuracy.

3. PROPOSED SYSTEM

we address the problem of data reduction for bug triage, i.e., how to reduce the bug data to save the labor cost of developers and improve the quality to facilitate the process of bug triage.

Data reduction for bug triage aims to build a small-scale and high-quality set of bug data by removing bug reports and words, which are redundant or non-informative.

In our work, we combine existing techniques of instance selection and feature selection to simultaneously reduce the bug dimension and the word dimension. The reduced bug data contain fewer bug reports and fewer words than the original bug data and provide similar information over the original bug data. We evaluate the reduced bug data according to two criteria: the scale of a data set and the accuracy of bug triage. In this paper, we propose a predictive model to determine the order of applying instance selection and feature selection. We refer to such determination as prediction for reduction orders. Drawn on the experiences in software metrics,1 we extract the attributes from historical bug data sets. Then, we train a binary classifier on bug data sets with extracted attributes and predict the order of applying instance selection and feature selection for a new bug data set.

ADVANTAGES OF PROPOSED SYSTEM

Experimental results show that applying the instance selection technique to the data set can reduce bug reports but the accuracy of bug triage may be decreased. Applying the feature selection technique can reduce words in the bug data and the accuracy can be increased. Meanwhile, combining both techniques can increase the accuracy, as well as reduce bug reports and words. Based on the attributes from historical bug data sets, our predictive model can provide the accuracy of 71.8 percent for predicting the reduction order.

We present the problem of data reduction for bug triage. This problem aims to augment the data set of bug triage in two aspects, namely a) to simultaneously reduce the scales of the bug dimension and the word dimension and b) to improve the accuracy of bug triage.

We propose a combination approach to addressing the problem of data reduction. This can be viewed as an application of instance selection and feature selection in bug repositories.

We build a binary classifier to predict the order of applying instance selection and feature selection. To our knowledge, the order of applying instance selection and feature selection has not been investigated in related domains

4. CONCLUSION

To locate a bug, developers use not only the content of the bug report but also domain knowledge relevant to the soft- ware project. We introduced a learning-to-rank approach that emulates the bug finding process employed by developers. The ranking model characterizes useful relationships between a bug report and source code files by leveraging domain knowledge, such as API specifications, the syntactic structure of code, or issue tracking data. Experimental evaluations on six Java projects show that our approach can locate the relevant files within the top 10 recommendations for over 70 percent of the bug reports in Eclipse Platform and Tomcat. Furthermore, the proposed ranking model out- performs three recent state-of-the-art approaches. Feature evaluation experiments employing greedy backward feature elimination demonstrate that all features are useful. When coupled with runtime analysis, the feature evaluation results can be utilized to select a subset of features in order to achieve a target trade-off between system accuracy and runtime complexity.

The proposed adaptive ranking approach is generally applicable to software projects for which there exists a sufficient amount of project specific knowledge, such as a comprehensive API documentation and an initial number of previously fixed bug reports . Further more, the ranking performance can benefit from informative bug reports and well documented code leading to a better lexical similarity and from source code files that already have a bug-fixing history

References

[1] G. Antoniol and Y.-G. Gueheneuc, "Feature identification: A novel approach and a case study," in *Proc. 21st IEEE Int. Conf. Softw. Maintenance*, Washington, DC, USA, 2005, pp. 357–366.

[2] G. Antoniol and Y.-G. Gueheneuc, "Feature identification: An epi- demiological metaphor," *IEEE Trans. Softw. Eng.*, vol. 32, no. 9, pp. 627–641, Sep. 2006.

[3] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "Debugadvisor: A recommender system for debugging," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, New York, NY, USA, 2009, pp. 373–382.

[4] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proc. Int. Conf. Softw. Eng.*, Piscataway, NJ, USA, 2013, pp. 712–721.

[5] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging usage similarity for effective retrieval of examples in code repositories," in *Proc. 18th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2010 pp. 157–166.

[6] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Looking for bugs in all the right places," in *Proc. Int. Symp. Softw. Testing Anal.*, New York, NY, USA, 2006, pp. 61–72.

[7] N. Bettenburg, S. Just, A. Schro€ter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2008, pp. 308–318.

[8] T. J. Biggerstaff, B. G. Mitbander, and D. Webster, "The concept assignment problem in program understanding," in *Proc. 15th Int. Conf. Softw. Eng.*, Los Alamitos, CA, USA, 1993, pp. 482–498.

[9] D. Binkley and D. Lawrie, "Learning to rank improves IR in SE," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Washington, DC, USA, 2014, pp. 441–445.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022 Mar. 2003.

[11] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in *Proc. ACM Conf. Comput. Supported Cooperative Work*, New York, NY, USA, 2010, pp. 301–310.

[12] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 3rd ed. Upper Saddle River, NJ, USA, Prentice-Hall, 2009.

[13] Y. Brun and M. D. Ernst, "Finding latent code errors via machine learning over program executions," in *Proc. 26th Int. Conf. Softw. Eng.*, Washington, DC, USA, 2004, pp. 480–490.

[14] M. Burger and A. Zeller, "Minimizing reproduction of software failures," in *Proc. Int. Symp. Softw. Testing Anal.*, New York, NY, USA, 2011 pp. 221–231.

[15] R. P. L. Buse and T. Zimmermann, "Information needs for soft- ware development analytics," in *Proc. Int. Conf. Softw. Eng.*, Piscat- away, NJ, USA, 2012, pp. 987–996.