Information Processing and Management 54 (2018) 1058–1076

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Leveraging textual properties of bug reports to localize relevant source files

Reza Gharibi*, Amir Hossein Rasekh, Mohammad Hadi Sadreddini, Seyed Mostafa Fakhrahmad*

Department of Computer Science & Engineering & IT, School of Electrical and Computer Engineering, Shiraz University, Shiraz, Iran

ARTICLE INFO

Keywords:
Bug localization
Bug report
Classification
Information retrieval
Textual analysis

ABSTRACT

Bug reports are an essential part of a software project's life cycle since resolving them improves the project's quality. When a new bug report is received, developers usually need to reproduce the bug and perform code review to locate the bug and assign it to be fixed. However, the huge number of bug reports and the increasing size of software projects make this process tedious and time-consuming. To solve this issue, bug localization techniques try to rank all the source files of a project with respect to how likely they are to contain a bug. This process reduces the search space of source files and helps developers to find relevant source files quicker. In this paper, we propose a multi-component bug localization approach that leverages different textual properties of bug reports and source files as well as the relations between previously fixed bug reports and a newly received one. Our approach uses information retrieval, textual matching, stack trace analysis, and multi-label classification to improve the performance of bug localization. We evaluate the performance of the proposed approach on three open source software projects (i.e., AspectJ, SWT, and ZXing) and the results show that it can rank appropriate source files for more than 52% of bugs by recommending only one source file and 78% by recommending ten files. It also improves the MRR and MAP values compared to several existing state-of-the-art bug localization approaches.

1. Introduction

A software bug is an error, flaw, failure or fault in a computer program that produces an incorrect or unexpected result. The quality of a software project is an important aspect for its success. Although different software quality control processes like software testing and software inspection are used, the increasing size of software and the limited development resources make developers release their software while still containing bugs. To deal with this problem, software projects set up bug tracking systems such as Bugzilla[1] and MantisBT[2] to keep track of bugs in their product effectively and then ask users all over the world to report bugs that they have confronted. However, this process can also create a new challenge for developers.

During the development and post-release life cycle of a software product, especially in a popular and well-known system, the number of bug reports can grow large and easily become tedious to handle. For example, there were 3,389 bug reports created for the

* Corresponding authors.
E-mail addresses: gharibi@cse.shirazu.ac.ir (R. Gharibi), ahrasekh@shirazu.ac.ir (A.H. Rasekh), sadredin@shirazu.ac.ir (M.H. Sadreddini), fakhrahmad@shirazu.ac.ir (S.M. Fakhrahmad).

[1] https://www.bugzilla.org/.
[2] https://www.mantisbt.org/.

https://doi.org/10.1016/j.ipm.2018.07.004
Received 9 February 2018; Received in revised form 14 July 2018; Accepted 15 July 2018

Eclipse platform product in 2013 alone (Ye, Bunescu, & Liu, 2014) and Mozilla project receives on average 300 new bug reports every day (Shokripour, Anvik, Kasirun, & Zamani, 2013). These numbers show that it's essential for the development teams to efficiently locate buggy files and resolve the problem.

After a bug is reported and confirmed, the development team should find which source files contain the problem and who will be assigned to make the code changes necessary to fix the fault (Jeong, Kim, & Zimmermann, 2009). As the software projects also become bigger and more complicated, one of the most time-consuming and difficult tasks is finding the buggy files that are responsible for a reported bug among hundreds or thousands of source files (Anvik, Hiew, & Murphy, 2005). Furthermore, developers who have to maintain and work on unfamiliar projects need to put a lot of effort in comprehending and understanding different parts of the program to find higher level concepts that are described by the bug report. Therefore, it would be beneficial to have an automatic system help us in this process that recommend buggy source files to developers based on the given bug report.

Researchers have proposed approaches to automate the process of finding the faulty source files. These approaches rank source files according to their relevance to the given bug report and make the developers' search space smaller. Developers then examine the files from the beginning of the ranked list to find the relevant faulty files.

One category of these approaches is spectrum-based that need the spectra information between different executions of the program to help locate faults (Abreu, Zoeteweij, Golsteijn, & van Gemund, 2009; Jones & Harrold, 2005; Le, Lo, Le Goues, & Grunske, 2016; Liu, Fei, Yan, Han, & Midkiff, 2006). Spectrum-based approaches usually need a test suite to execute the program and collect its execution information; while the lack of test cases in software projects makes it hard to collect many correct and failing execution traces. Another category of bug localization approaches is information retrieval (IR)-based techniques that are static and don't require runtime information of the program (Rahman, Ganguly, & Sakib, 2015; Saha, Lease, Khurshid, & Perry, 2013; Wang & Lo, 2014; Wong et al., 2014; Zhou, Tong, Chen, & Han, 2017; Zhou, Zhang, & Lo, 2012). These approaches locate the bug-relevant files based on a given bug report. In fact, the text in the summary and description of a given bug report is considered as a query and source files are documents that are ranked based on their relevance to the bug report's content.

As shown in Fig. 1, there are similar word tokens in different parts of a bug report[3] and its associated source file. The intuition here is that software elements that share many common word tokens with the given bug report are likely to be relevant to the bug and can help locate the appropriate source file for each bug report quicker (Zhou et al., 2012). Apart from word tokens, full identifiers are often present in bug reports and can further help in going toward the right source file (Saha et al., 2013). Some matching positions in the source file are more important than others. We can give more weight to matching terms or word tokens in these positions in order to avoid their effect loss in finding the relevant source file. Words in bug reports also have different importance. For example, this can be seen in Fig. 2 as words with specific part-of-speech or POS tags are more important, and mostly nouns and verbs are the common words between the bug report and source code.

Some bug reports contain a stack trace and the relevant file name for a bug report can be found there. It has been shown that the relevant source file of about 90% of bugs is located within the top 10 stack frames (Schröter, Bettenburg, & Premraj, 2010), and as expected, most of the times it is in the first frame. Fig. 3 shows an example of a bug report[4] containing a stack trace of a component that has crashed. The relevant source file for this bug report is "ProgramElement.java", which is located in the first frame of the stack trace.

Sometimes the terms used to describe the bug in the bug report are not the same as the ones used in the source code, but if we consider them semantically, they are close in meaning. For example in Fig. 4, the word "camera" is used, and the related source file to this bug report has the term "capture" in its name. We know that "camera" and "capture" are related to each other; therefore we can use semantic similarity to overcome this lexical mismatch between the bug report and source file. There are many different mismatches in what bug reports contain and what developers consider useful in them to better localize the bugs (Bettenburg et al., 2008), but lexical mismatch makes it harder for a bug localization system. In this case, some domain knowledge about the software project is needed to locate relevant files (Ye et al., 2014).

Previously fixed source files may be responsible for newer similar bugs (Murphy-Hill, Zimmermann, Bird, & Nagappan, 2013), and sometimes people submit bug reports similar to these already fixed bug reports that may affect the same source file. Since usually there are already fixed bug reports available in bug tracking systems of software projects, common tokens shared between new and fixed reports can be used to further improve the bug localization performance.

Although many bug localization approaches are proposed, there are still some gaps that can be covered to improve their accuracy. Motivated by these observations, in this paper, we propose a five-component bug localization approach that combines various textual aspects of bug reports and source files to locate relevant source files for each bug report. The components consist of a token matching component that tries to find exact matches of some specific textual tokens of bug reports in specific positions of source files; a Vector Space Model (VSM) similarity component that utilizes a variation of the revised Vector Space Model (rVSM) (Zhou et al., 2012) with specifically extracted word tokens to better score source files for a bug report; a stack trace component that uses stack traces in bug reports if available; a semantic similarity component, and finally a fixed bug reports component that uses previously fixed bug reports and a multi-label classification algorithm to score suspicious faulty source files according to existing solved bug reports.

As the main contribution of this paper, we first develop these five components to get separate ranking scores for each source file. Then we combine these scores to get a final ranking score and use that to rank relevant source files with respect to each bug report. After that, we try to show how good our approach works on the bug reports of real-world software projects. Therefore, we evaluate

[3] https://bugs.eclipse.org/bugs/show_bug.cgi?id=80720.
[4] https://bugs.eclipse.org/bugs/show_bug.cgi?id=43709.