Information and Software Technology 82 (2017) 177–192

Contents lists available at ScienceDirect

Information and Software Technology

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

Improved bug localization based on code change histories and bug reports[☆]

Klaus Changsun Youm[a,b,*], June Ahn[a], Eunseok Lee[a,*]

[a] Department of Information and Communication Engineering, Sungkyunkwan University, Suwon, Republic of Korea
[b] Mobile Communication and Business, Samsung Electronics, Suwon, Republic of Korea

ARTICLE INFO

Article history:
Received 13 March 2016
Revised 1 November 2016
Accepted 1 November 2016
Available online 9 November 2016

Keywords:
Bug localization
Information retrieval
Bug reports
Stack traces
Code change history
Method analysis

ABSTRACT

Context: Several issues or defects in released software during the maintenance phase are reported to the development team. It is costly and time-consuming for developers to precisely localize bugs. Bug reports and the code change history are frequently used and provide information for identifying fault locations during the software maintenance phase.
Objective: It is difficult to standardize the style of bug reports written in natural languages to improve the accuracy of bug localization. The objective of this paper is to propose an effective information retrieval-based bug localization method to find suspicious files and methods for resolving bugs.
Method: In this paper, we propose a novel information retrieval-based bug localization approach, termed Bug Localization using Integrated Analysis (BLIA). Our proposed BLIA integrates analyzed data by utilizing texts, stack traces and comments in bug reports, structured information of source files, and the source code change history. We improved the granularity of bug localization from the file level to the method level by extending previous bug repository data.
Results: We evaluated the effectiveness of our approach based on experiments using three open-source projects, namely AspectJ, SWT, and ZXing. In terms of the mean average precision, on average our approach improves the metric of BugLocator, BLUiR, BRTracer, AmaLgam and the preliminary version of BLIA by 54%, 42%, 30%, 25% and 15%, respectively, at the file level of bug localization.
Conclusion: Compared with prior tools, the results showed that BLIA outperforms these other methods. We analyzed the influence of each score of BLIA from various combinations based on the analyzed information. Our proposed enhancement significantly improved the accuracy. To improve the granularity level of bug localization, a new approach at the method level is proposed and its potential is evaluated.

1. Introduction

Software maintenance costs after the release of a product are greater than the cost of the design and implementation phases [2–4]. Issues or defects in the released software are updated and managed during the development phase using the bug or issue management system. Developers who are assigned to resolve the report initiate activities to fix the problem. They attempt to reproduce the same result in the bug report and then find the locations of the defects. However, it is costly and time-consuming to precisely localize bugs, and bug localization is a tedious process for developers. In cases of large software products, it is difficult for hundreds of developers to resolve the large number of bug reports. Therefore, effective methods for locating bugs automatically from bug reports are desirable in order to reduce the resolution time and software maintenance costs.

To this end, various studies regarding Change Impact Analysis (CIA), based on differences in the analysis of source file versions, were proposed during the 2000s [5–12]. In the 2010s, numerous Spectrum-Based Fault Localization (SBFL) techniques have been suggested and evaluated [12–16]. Recently, researchers have applied various Information Retrieval (IR) techniques [17], which are mainly used to search the text domain for software maintenance for the purposes of feature location, developer identification and impact analysis [18]. IR-based bug localization techniques have attracted significant attention due to their relatively low computational cost and improved accuracy compared to change impact analysis or spectrum-based fault localization. In these IR approaches, the main idea is that a bug report is treated as a query and that the source files in the software product to be searched

☆ A preliminary version of this paper was presented and best paper awarded at the 22th Asia-Pacific Software Engineering Conference (APSEC 2015) [1].
* Corresponding authors.
E-mail addresses: klausyoum@skku.edu, klaus.youm@samsung.com (K.C. Youm), ahnjune@skku.edu (J. Ahn), leees@skku.edu (E. Lee).

http://dx.doi.org/10.1016/j.infsof.2016.11.002
0950-5849/© 2016 Elsevier B.V. All rights reserved.

comprise the document collection. To improve the accuracy of locating bugs, the following are all used in various combinations: similarity analysis of previously fixed bugs [19], the use of structured information of the source files instead of treating them as simple documents [20], version change history analysis [21] and stack trace analysis [22–24].

We analyzed the bug/issue management process to identify the pieces of information useful to developers for localizing bugs. With these considerations in mind, we propose a static and integrated analysis approach for bug localization by utilizing texts, stack traces and comments in bug reports, structured information of the source files, and the source code change history. First, we analyze the similarity between texts in a bug report and the source files using an IR method. We adopt the revised Vector Space Model (rVSM) of Zhou et al. [19], then integrate the structured information analysis of source files, the effectiveness of which was demonstrated by Saha et al. [20]. Second, the similarities of previously fixed bug reports are analyzed using a basic IR method [19]. Third, if a bug report includes stack traces, stack traces are also analyzed to identify suspicious files therein [23]. Fourth, an analysis of the historical information from the source code changes is performed to identify suspicious files and methods for predicting bugs [21]. Fifth, we integrate the above four types of information to localize suspicious files in each bug report; this represents an output of bug localization at the file level. Sixth, from the ranked suspicious files, we analyze the similarity between methods in the files and bug reports. Finally, suspicious methods are ranked and listed based on a combination of the scores of the fourth and sixth steps. This also represents an output of bug localization at the method level.

The contributions of our research are as follows:

1. We propose a novel IR-based bug localization approach termed Bug Localization using Integrated Analysis (BLIA). We utilize the content, stack traces and comments in bug reports, structured information of the source files, and the source code change history. We design a combined method to integrate all analyzed data in order to localize bugs at not only the file level, but also the method level. BLIA v1.0, which is a preliminary version, is focused on bug localization at the file level. The current version, BLIA v1.5, extends its implementation to both the method level and the analysis of the comments in bug reports.
2. We find the optimized range of parameters that control the influence rate of each piece of information analyzed.
3. Bug repositories for our experiments were extended. The comments, fixed methods and fixed commits for each bug report were inserted. The data are available to improve the accuracy of bug localization research.

The remainder of this paper is organized as follows. Section 2 describes the background for this work. Section 3 describes our proposed BLIA approach. In Section 4, we present our experimental design and evaluation metrics. We discuss the experimental results and threats to validity in Section 5 and Section 6, respectively. Section 7 surveys related work. Finally, our conclusion and directions for future work are presented in Section 8.

2. Background

In this section, we first present a bug/issue management process to understand how bug reports are resolved. With an understanding of this process, we can identify factors that are helpful in locating bugs. We describe an example of an actual bug report from an open-source project. We demonstrate the general process and techniques of IR-based bug localization. In the last subsection, we provide historical information from the software configuration repository.

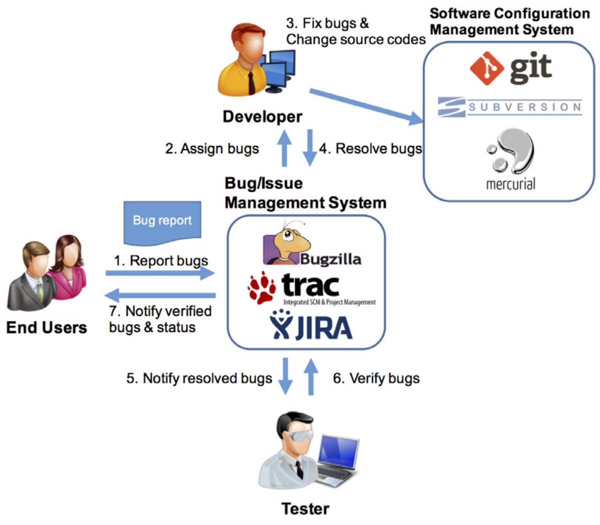
3. Fix bugs & Change source codes
Software Configuration Management System
Developer
2. Assign bugs
4. Resolve bugs
Bug/Issue Management System
Bug report
1. Report bugs
7. Notify verified bugs & status
End Users
5. Notify resolved bugs
6. Verify bugs
Tester

Fig. 1. Bug/issue management process.

2.1. Bug/issue management process

Fig. 1 illustrates a bug/issue management process used by developers to resolve a bug report after its assignment. Users submit a report to a bug/issue management system when they use a released software product and unexpected errors occur. Not only do industrial projects apply bug/issue management systems such as Bugzilla, Trac and JIRA, but open-source projects do as well. A bug report is assigned to developers who attempt to find the location of the bug. They usually try to reproduce the bug scenario in the bug report. The developers who find the location and root cause of a defect first fix the source files and then test the modified software again. If the error scenario in the bug report does not recur, the status of the bug report is changed to “Resolved” by the assigned developers. After resolution of the bug report, if the development team has a separate testing team, a tester is notified of the updated status. The tester checks the scenario of the reported bug, and then verifies that the bug has been resolved. The status is changed to “Verified” if the testing is completed without errors. A verification notification of the bug report is then sent to the reporter.

2.2. Bug reports

Bug reports are vital clues for developers for finding the location of defects. Developers guess which source files and lines are buggy after understanding the assigned bug report. Usually, they attempt to reproduce the scenario in the bug report, and then attempt to trace the root cause of the defects and find the location of the source files.

Table 1 presents an existing bug report[1] from Eclipse Bugzilla[2]. A bug report comprises the bug ID, summary, description, reported date, fixed date, status and other related fields. Some bug reports include stack traces of when the exception occurred. This stack trace information is critical for localizing a bug [25]. For example, developers attempt to reproduce an issue scenario and assume that the location of a bug is based on class names or method names in the bug report. Comments included in the bug report provide further information on the error situation.

Furthermore, developers search for similar bug reports in the bug repository. If similar bug reports are found, the fixed files used

[1] https://goo.gl/POC65q.
[2] https://bugs.eclipse.org/bugs/.