Exploring Metadata in Bug Reports for Bug Localization

Xiaofei Zhang, Yuan Yao, Yaojing Wang, Feng Xu, Jian Lu
National Key Laboratory for Novel Software Technology, Nanjing University
Collaborative Innovation Center for Novel Software Technology and Industrialization
Nanjing 210023, China
xiaofeizhang@smail.nju.edu.cn, y.yao@nju.edu.cn, wyj@smail.nju.edu.cn, {xf, lj}@nju.edu.cn

***Abstract*—Information retrieval methods have been proposed to help developers locate related buggy source files for a given bug report. The basic assumption of these methods is that the bug description in a bug report should be textually similar to its buggy source files. However, the metadata (such as the component and version information) in bug reports is largely ignored by these methods. In this paper, we propose to explore the metadata for the bug localization task. In particular, we first apply a generative model to locate buggy source files based on the bug descriptions, and then propose to add the available metadata in bug reports into the localization process. Experimental evaluations on several software projects indicate that the metadata is useful to improve the localization accuracy and that the proposed bug localization method outperforms several existing methods.**

***Index Terms*—Bug localization, bug reports, bug metadata, supervised topic modeling**

I. Introduction

In many software projects, the bug tracking system has been introduced to collect and manage bug reports. When a tester or developer discovers an abnormal behavior of the software, he or she will be asked to fill in a form provided by the bug tracking system. The form consists of a bug description and some bug metadata such as product, component, version, and hardware (see Fig. 1 as an example). Such a bug tracking system is helpful for developers to maintain software quality [1].

Even with a bug tracking system, it is still not an easy task to locate the buggy source files. For example, a developer who is assigned to deal with a bug report probably needs to analyze the information of the bug report, reproduce the abnormal behavior [2], and perform code review [3] to find the cause. To save the developers' effort, a better way is to automatically recommend a list of potential buggy source files for a given bug report. This task is referred to as bug localization.

For the bug localization task, information retrieval (IR) methods have been widely used. The IR methods treat each bug report as a query and the existing source files in the code repository as a collection of documents. Then, the bug localization task becomes a standard IR problem where the goal is to find textually similar source files for a given bug report. Under the IR framework, the key difference of the existing methods is on the similarity computation aspect. For example, Lukins et al. [4] applied Latent Dirichlet Allocation (LDA) [5] to learn the latent topics of bug reports and source files, based on which the similarity can be computed. Zhou et al. [6] adopted the vector space model (VSM) to compute similarities and further incorporated the similarities between bug reports to improve performance. Recently, Lam et al. [7] and Huo et al. [8] used deep learning techniques to extract features for bug reports and sources files, and compute similarities on these features.

One limitation of the existing IR methods is that they tend to ignore the metadata in the bug reports. On the one hand, most of the existing IR methods use only the bug descriptions in the bug reports (e.g., [6], [8]–[10]); even for the few methods that use metadata (e.g., [11]), they simply treat the metadata as part of the bug description. On the other hand, metadata in bug reports is potentially useful for bug localization as it contains valuable information (such as the component and version information) in addition to the bug description; for example, the component metadata may indicate that a subset of source files are related to the bug report; it has also been shown that metadata is helpful for other tasks such as bug report identification [12].

In this paper, we propose to systematically explore the metadata for the bug localization task. In particular, instead of following the existing IR methods, we first use a supervised topic modeling method to locate buggy source files based on the bug descriptions. The intuition is that compared to unsupervised IR methods, supervised methods may have better performance in terms of accurately identifying the relevant source files. The basic idea to formulate the problem as a supervised learning problem is by using existing bug fixing histories as supervision information and use the source filenames as the supervision labels. Further, we discover that the naming of source files often follow a hierarchy structure (e.g., the full name of a source file is the concatenation of the module name and the class name) and the substrings of filenames often appear multiple times in the content of bug report descriptions. For example, 'aspectj/weaver/bcel/LazyClassGen.java' is a source filename and its substrings like 'weaver' and 'lazyclass' may frequently appear in the descriptions of the related bug reports. We propose to incorporate this phenomenon into topic modeling, and we name this model as L2SS (Label-to-SubString).

Next, we propose to incorporate several types of metadata into L2SS. In particular, we consider seven types of metadata and add each of them into L2SS to see its usefulness. The metadata consists of component, version, platform, operating system, priority, severity, and reporter. The basic idea to incorporate metadata is to model it as the prior probability for choosing topics. For example, if the component metadata is ‘UI’, it is probable that the buggy source files are related to the UI. Such information can be used to lead the search direction of possible buggy source files for a given bug report. The model with metadata incorporated is named as L2SS+X where ‘X’ stands for the corresponding matadata.

To verify the effectiveness of the proposed method, we conduct experiments on four open source projects with a total of 36,417 bugs. The results show that the proposed method outperforms several existing methods in terms of localization accuracy, and that the metadata is useful to further improve the localization accuracy. For example, when evaluated on the JDT (Java development tools) dataset (which contains 5,211 fixed bug reports), one of the proposed method (i.e., L2SS+CM) is able to find the related buggy files within the top 10 recommendations for over 70% bug reports, while the BugLocator competitor [6] can help 55% bug reports within the top 10 recommendations. Meanwhile, the proposed method can be efficiently pre-trained and used at the prediction stage.

The main contributions of this paper include:

1) We propose a supervised topic modeling method L2SS for the bug localization task. L2SS uses existing bug fixings and treats the source filenames as the supervision information. By doing so, L2SS can be applied when developers only have access to the source filenames.
2) We study the usefulness of several types of metadata in the bug report, and revise L2SS to incorporate these metadata (denoted as L2SS+). For example, when the component metadata is incorporated, we have the L2SS+CM method.
3) We evaluate the performance of L2SS and L2SS+ on several real datasets, and the results show the effectiveness and efficiency of the proposed methods. For example, L2SS+CM can achieve up to 16.9% improvement compared to its best existing competitors.

The remainder of this paper is organized as follows. Section II provides some background information. Section III presents the proposed bug localization method. Section IV describes the experimental setup and the research questions. Section V presents the experimental results. Section VI discusses the threats to validity. Section VII covers related work, and Section VIII concludes.

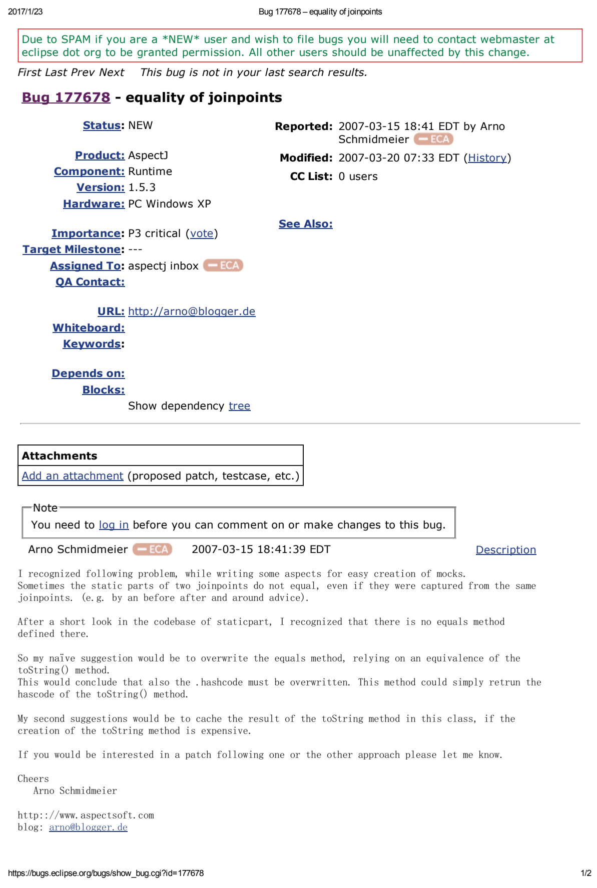
II. Background

In this section, we introduce some background information.

A. Bug Reports and Metadata

Bug tracking system has been widely used in software projects. Take the Bugzilla System in Eclipse project as an example. Fig. 1 shows a real bug report[1] (ID:177678) for AspectJ in Eclipse.

[1]https://bugs.eclipse.org/bugs/show_bug.cgi?id=177678



2017/1/23 Bug 177678 – equality of joinpoints

Due to SPAM if you are a *NEW* user and wish to file bugs you will need to contact webmaster at eclipse dot org to be granted permission. All other users should be unaffected by this change.

First Last Prev Next This bug is not in your last search results.

Bug 177678 - equality of joinpoints

Status:	NEW
Product:	AspectJ
Component:	Runtime
Version:	1.5.3
Hardware:	PC Windows XP
Importance:	P3 critical (vote)
Target Milestone:	---
Assigned To:	aspectj inbox ECA
QA Contact:	
URL:	http://arno@blogger.de
Whiteboard:	
Keywords:	
Depends on:	
Blocks:	
	Show dependency tree
Reported:	2007-03-15 18:41 EDT by Arno Schmidmeier ECA
Modified:	2007-03-20 07:33 EDT (History)
CC List:	0 users
See Also:	

Attachments
Add an attachment (proposed patch, testcase, etc.)

Note
You need to log in before you can comment on or make changes to this bug.

Arno Schmidmeier ECA 2007-03-15 18:41:39 EDT Description

```
I recognized following problem, while writing some aspects for easy creation of mocks.
Sometimes the static parts of two joinpoints do not equal, even if they were captured from the same
joinpoints. (e.g. by an before after and around advice).

After a short look in the codebase of staticpart, I recognized that there is no equals method
defined there.

So my naïve suggestion would be to overwrite the equals method, relying on an equivalence of the
toString() method.
This would conclude that also the .hashcode must be overwritten. This method could simply retrun the
hascode of the toString() method.

My second suggestions would be to cache the result of the toString method in this class, if the
creation of the toString method is expensive.

If you would be interested in a patch following one or the other approach please let me know.

Cheers
   Arno Schmidmeier

http:://www.aspectsoft.com
blog: arno@blogger.de
```

https://bugs.eclipse.org/bugs/show_bug.cgi?id=177678 1/2

Fig. 1. A bug report example in the AspectJ project.

We can see from Fig. 1 that a bug report typically contains description, status, product, component, version, hardware (platform and operating system), importance (priority and severity), etc. Except for the bug description, we refer to the rest information as metadata. The metadata is widely ignored by existing information retrieval methods, while it may be helpful for bug localization. For example, the component metadata may indicate a subset of source files that are related to the bug; the bug reporter metadata can also play similar roles as a reporter may be responsible for testing a specific part of the code repository.

B. Supervised Topic Modeling

Topic modeling (e.g. LDA [5]) has been used to capture the topical similarities between bug reports and source codes to locate bugs [4]. To make use of the existing fixing histories between source files and bug reports, a natural tool is supervised topic modeling. In bug localization task, bug reports can be seen as documents and the related source filenames can be seen as the labels for the documents. Then, given a bug report, the goal is to predict its related labels (i.e., source filenames).

Fig. 2 shows an example (LLDA [13]) of supervised topic modeling. The basic idea of LLDA is as follows. First, each