It's Not a Bug, It's a Feature: Does Misclassification Affect Bug Localization?

Pavneet Singh Kochhar, Tien-Duy B. Le, and David Lo
School of Information Systems
Singapore Management University
{kochharps.2012,btdle.2012,davidlo}@smu.edu.sg

ABSTRACT

Bug localization refers to the task of automatically processing bug reports to locate source code files that are responsible for the bugs. Many bug localization techniques have been proposed in the literature. These techniques are often evaluated on issue reports that are marked as bugs by their reporters in issue tracking systems. However, recent findings by Herzig et al. find that a substantial number of issue reports marked as bugs, are not bugs but other kinds of issues like refactorings, request for enhancement, documentation changes, test case creation, and so on. Herzig et al. report that these misclassifications affect bug prediction, namely the task of predicting which files are likely to be buggy in the future. In this work, we investigate whether these misclassifications also affect bug localization. To do so, we analyze issue reports that have been manually categorized by Herzig et al. and apply a bug localization technique to recover a ranked list of candidate buggy files for each issue report. We then evaluate whether the quality of ranked lists of *reports reported as bugs* is the same as that of *real bug reports*. Our findings shed light that there is a need for additional cleaning steps to be performed on issue reports before they are used to evaluate bug localization techniques.

Categories and Subject Descriptors

K.6.3 [**Software Management**]: Software Maintenance

General Terms

Experimentation

Keywords

Misclassification, Bug Localization

1. INTRODUCTION

To improve the quality of software systems, developers often open an issue tracking system for people to submit issue reports. An issue report can contain a description of a bug, a new feature request, a request for re-documentation, a request for additional test cases, and so on. One of the most important issue reports are bug reports since these bugs can affect the reliability of software systems. For large systems, the number of bug reports, could be too large for developers to handle. A Mozilla developer, as quoted by Anvik et al., mentions that "Everyday, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" [1]. Thus, there is a need for techniques that can help developers resolve bug reports faster. The core part of this activity would be the identification of buggy files that are responsible for the bugs.

To address the above need, researchers have proposed techniques that take as input a bug report and use information within the bug report to identify source code files that are likely to be related to the report. These techniques are often referred to as bug localization and many such techniques have been proposed recently, e.g., [7, 10, 13, 11]. Among these techniques, standard information retrieval based techniques that compute the similarity between the textual description of a bug report and textual description of a source code file are often employed with relative success. For each bug report, these techniques return a ranked list of source code files sorted based on their likelihood to be the ones responsible for the bug. To evaluate these techniques, often historical closed and fixed issue reports marked as bugs are collected from issue tracking systems. The goal of the evaluation is to see if the ranked list of files returned by bug localization technique contains actual buggy files that need to be modified to resolve the bug report.

Recently, Herzig et al. have manually analyzed more than 7,000 issue reports from the issue tracking systems of 5 software projects: HTTPClient, Jackrabbit, Lucene-Java, Rhino, and Tomcat5 [5]. They reported that more than 40% of the issue reports are wrongly classified. About every 1 out of 3 issue reports that are marked as bugs are not bugs. They have also shown that misclassification affects defect prediction which is the task to predict which files are likely to be buggy in the future. In this study, we extend Herzig et al.'s work by asking this question: does misclassification affect bug localization? Similar with Herzig et al.'s work our goal is to investigate if there is a threat that affects the validity of the evaluations performed by studies on bug localization including those of our own, e.g., [13].

To investigate whether misclassification affects bug localization, we reuse the manually categorized bug report datasets of Herzig et al. which is made publicly avail-

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597105

able at: `http://www.st.cs.uni-saarland.de/softevo/bugclassify/`. Herzig et al.'s datasets contain bug reports from Jira and Bugzilla issue reports. Past studies have shown that many bug reports in Bugzilla are poorly linked [3, 2, 12], however, bug reports in JIRA are often well linked due to the fact JIRA provides add-ons that can connect issues to commits in version control systems [4]. Thus, we only use Herzig et al.'s bug report datasets that originate from a JIRA issue tracking system. These include 5,591 bug reports from HTTPClient's, Jackrabbit's, and Lucene-Java's JIRA issue tracking systems. From these bug reports, we can compare the accuracy of a bug localization technique on issue reports that are marked as bugs by their reporters (`reported`) to the accuracy of the bug localization technique on those that are actual bug reports (`actual`).

In our study, we use a bug localization technique based on vector space model (VSM), which is a standard information retrieval method. This technique has been shown to outperform many other information retrieval methods in a past study by Rao and Kak [10]. Extended vector space models have also been used by state-of-the-art bug localization techniques [13, 11]. In this work, we choose to use the original VSM as if the performance of this base technique is affected due to misclassification, other techniques derived from it would also likely to be affected. To measure the performance of a bug localization technique, we make use of average precision which is a standard information retrieval metric, and it has also been used in past bug localization studies, e.g., [10, 13, 11].

In this work, we would like to answer three research questions:

RQ1 Does the performance of a bug localization technique differs for issue reports reported as bugs and actual bug reports?

RQ2 Which misclassification types have more impact to a bug localization technique?

RQ3 What can we do to mitigate the impact of misclassification to a bug localization technique?

The contributions of this paper are as follows:

1. We are the first to analyze the impact of misclassification on bug localization. Our work extends Herzig et al.'s work that study the impact of misclassification on bug prediction. Bug localization and bug prediction are related but differ in terms of the inputs considered (bug reports vs. history) and task solved (predict specific buggy files responsible for a report vs. predict future buggy files).
2. We analyze the types of misclassifications that have more impact on the performance of bug localization techniques and suggest what can be done to mitigate the effects of these misclassifications.

The structure of this paper is as follows. In Section 2, we describe preliminary information on issue reports, text pre-processing, and a bug localization technique that we use in this paper. In Section 3, we present our empirical study methodology. In Section 4, we present our empirical study results. We discuss related work in Section 5. We finally conclude and mention future work in Section 6.

2. PRELIMINARIES

Issue Reports: An issue report contains a number of fields and each of them carries a piece of information. In this work, three fields are particularly interesting to us: (1) short summary, (2) longer description, and (3) issue category.

Text Preprocessing: Text preprocessing is an essential task in information retrieval based bug localization techniques. There are three text preprocessing steps: text normalization, stop word removal, and stemming. These steps have been performed by past bug localization studies, e.g., [13]. At the end of these preprocessing steps, each bug report and source code file is represented as a bag (i.e., multi-set) of words.

In the text normalization step, we remove punctuation marks, special symbols, and number literals from bug reports, and source code files. To normalize source code files, we utilize JDT library[1] to extract Abstract Syntax Trees (ASTs) from source code. We keep texts in AST nodes corresponding to identifiers and string literals. Furthermore, we split identifiers into word tokens following Camel casing convention. In the stop word removal step, we remove from bug reports commonly occurring English words (e.g., "I", "you", "we", "are", etc.); we use the stop word list from: `http://dev.mysql.com/doc/refman/5.6/en/fulltext-stopwords.html`. We also remove programming language keywords from source code files (e.g., *public*, *class*, *if*, *for*, etc.). We remove these stop words as they carry little meaning. Next, we convert all word tokens into lower case. In the stemming step, we reduce a word to its root form. For example, we reduce "mapping", "mapped", and "maps" to "map". To do this, we apply the well known Porter Stemming Algorithm and use its implementation made publicly available at: `http://tartarus.org/martin/PorterStemmer/`.

Bug Localization Using VSM: We implement a bug localization technique that is based on vector space model (VSM). It takes as input a query (i.e., preprocessed bug report), and a corpus of documents (i.e., preprocessed source code files) and outputs a ranked list of documents sorted by their textual similarity to the input query. VSM represents a document as a vector of weights, where each weight corresponds to a word in the document. The weight of each word is usually computed using the product of its term frequency and its inverse document frequency, following the standard tf-idf weighting scheme [9]. The following is the tf-idf weight of word w in document d given a set of documents D (denoted as *tf-idf*(w, d, D)):

$$\textit{tf-idf}(w, d, D) = \log(f(w, d) + 1) \times \log \frac{\mid D \mid}{\mid \{d_i \in D : w \in d_i\} \mid}$$

In the above equation, $f(w, d)$ is the number of times word w occurs in document d, and $w \in d_i$ denotes that word w appears in document d_i. Textual similarity between query q and document d is obtained by computing the cosine similarity between the two vectors representing q and d [9].

Evaluation Metric: We evaluate bug localization techniques using *average precision* [9], which is a widely used metric in information retrieval and it has been used to evaluate many past bug localization techniques [10, 13, 11]. A bug localization technique outputs a ranked list of source code files given an input bug report. Given a ranked list for

[1] http://www.eclipse.org/jdt/

a bug report r, we can compute an average precision score as follows:

$$AvgP(r) = \frac{\sum_{k=1}^{D_r} P@k \times rel(k)}{\text{Total number of relevant files}}$$

$$P@k = \frac{\#\text{ relevant files in the top-k elements}}{k}$$

In the above equations, D_r is the size of the ranked list; $P@k$ is the precision at k, which is the ratio between the number of relevant files (i.e., source code files that need to be modified to resolve bug report r) among the top-k elements in the ranked list; $rel(k)$ is a function that returns 1 if the file at position k is relevant (i.e., it needs to be modified to resolve the bug), and 0 otherwise.

3. STUDY APPROACH

Our empirical study methodology consists of several steps: data acquisition, bug localization, effectiveness measurement, and statistical test. We describe each of these steps in the following paragraphs.

Data Acquisition: We download Herzig et al.'s datasets which include the identifiers of issue reports that they have manually analyzed from `www.st.cs.uni-saarland.de/softevo/bugclassify`. We download the actual issue reports from the corresponding JIRA repositories. We extract the textual contents of the summary and description fields of these issue reports and perform the preprocessing steps described in the previous section. After this step, each bug report is represented as a bag-of-words. We also analyze history data from version control systems to find the commits that are linked to the various issue reports. In JIRA, it is easy to identify these commits as the identifier of an issue report would appear as the first few characters in a commit log. Given these commits, we also extract the source code files in the code base prior to the commits that address the issue, and the set of source code files that are modified to resolve the issue. For each source code file in the code base, we extract words that appear in the file and perform the preprocessing steps described in the previous section. After this step, each source code file is represented as a bag-of-words.

Bug Localization: At the end of the data acquisition step, we have for each issue report, the textual content of the issue reports, the textual content of each source code file in the code base prior to the fix, and a set of source code files that are modified in the commits that resolve the issue report. We input the textual content of the issue reports and source code files to a bug localization technique which would output a ranked list of files in the code base sorted based on their similarity to the bug report. We use the VSM-based bug localization technique presented in the previous section.

Effectiveness Measurement: At the end of the bug localization step, we have for each issue report, a ranked list of source code files. We also have the set of files that are modified to address the issue report – which we treat as the ground truth. We compare the ranked list with the ground truth and for each report, we compute the average precision score presented in the previous section.

Statistical Test: At the end of the previous step, we have a set of average precision scores. Note that each dataset contains issue reports that are marked by their reporters as bugs (`Reported`), and issue reports that are actual bug reports, i.e., they are labeled by Herzig et al. as bugs (`Actual`). We extract a set of average precision scores for `Reported` and `Actual`. We then compute the mean of these scores and perform the well-known Mann-Whitney U test [8] to see if the differences in the means are significant.

4. STUDY RESULTS

In the following paragraphs we describe the answers to the research questions that we pose in the introduction section.

RQ1-Effect of Misclassification on Bug Localization: The Mean Average Precision (MAP) scores for the two cases: reports marked as bugs (`Reported`), and actual bug reports (`Actual`) are shown in Table 1. We can note that there is a -2.33%–12.25% difference in terms of the MAP scores. We have also performed a Mann-Whitney U test at 0.05 level of significance and we find that the differences are significant. Thus, misclassification significantly affects bug localization results.

Table 1: Mean Average Precision (MAP) Scores for `Reported` and `Actual`

Project	`Reported`	`Actual`	Difference
HTTPClient	0.429	0.419	-2.33%
Jackrabbit	0.302	0.339	12.25%
Lucene-Java	0.301	0.322	6.98%

RQ2-Effect of Different Misclassification Types: To answer this research question, we modify our study methodology slightly by omitting issue reports that suffer from a misclassification type (e.g., `RFE` misclassified as `BUG` (`RFE` to `BUG`)), one type at a time and recalculate the MAP scores. We would like to find the misclassification type that has the most impact to the difference in the MAP scores for `Reported` and `Actual`. In Herzig et al.'s dataset, they identify 13 different categories of issue reports: `BUG`, `RFE`, `IMPROVEMENT`, `DOCUMENTATION`, `REFACTORING`, `BACKPORT`, `CLEANUP`, `SPEC`, `TASK`, `TEST`, `BUILD_SYSTEM`, `DESIGN_DEFECT`, and `OTHERS` [2]. The results are shown in Table 2. We note that the misclassification types with the most impact are `TEST` to `BUG` and `IMPROVEMENT` to `BUG`.

RQ3-Mitigation Strategy: To suggest what can be done to mitigate the impact of misclassification on bug localization, we analyze `TEST` to `BUG` and `IMPROVEMENT` to `BUG` misclassification cases which have been shown in the answer to RQ2 to be the dominant causes of the performance difference. We find that many these cases correspond to bug reports where no source code files are modified (`TEST` to `BUG`), and bug reports whose summary or description fields explicitly specify the buggy files (`IMPROVEMENT` to `BUG`). For example, the summary field of bug report HTTPCLIENT-1036 is "StringBody has incorrect default for characterset" and the relevant file is StringBody.java.

Thus, to mitigate effect of misclassification in the evaluation of bug localization technique, we can omit all bug reports where no source code files are modified to resolve them, or whose summary or description already explicitly specify the buggy files. For the latter case, there is no need

[2]In their paper, 6 categories are identified. We use the categories in their publicly downloadable dataset which include 14 categories. We merge `UNKNOWN` to `OTHERS`.

Table 2: Mean Average Precision (MAP) Scores when Issue Reports of a Particular Misclassification Type are Omitted. Omit. = Omitted, Misclass. = Misclassification, HC = HTTPClient, JB = Jackrabbit, LJ = Lucene-Java. The last column is the MAP of all three projects.

Omit. Misclass. Type (Actual to Reported)	HC	JB	LJ	Overall
None	0.429	0.302	0.301	0.312
RFE to BUG	0.427	0.303	0.304	0.313
DOCUMENTATION to BUG	0.43	0.304	0.305	0.315
IMPROVEMENT to BUG	0.416	0.299	0.295	0.307
REFACTORING to BUG	0.428	0.301	0.301	0.311
BACKPORT to BUG	0.43	0.303	0.300	0.313
CLEANUP to BUG	0.429	0.303	0.303	0.314
SPEC to BUG	0.435	0.302	0.303	0.312
TASK to BUG	0.432	0.302	0.301	0.312
TEST to BUG	0.429	0.328	0.313	0.334
BUILD_SYSTEM to BUG	0.429	0.306	0.303	0.315
DESIGN_DEFECT to BUG	0.424	0.301	0.301	0.311
OTHERS to BUG	0.439	0.303	0.301	0.313

for bug localization as developers can localize the buggy files by just reading the reports.

Threats to Validity: The validity of our study is largely dependent on the accuracy of the category labels that are created by Herzig et al. during their manual investigation of the issue reports which is a threat to internal validity. Also, we have only investigated 5,591 issue reports from three projects which is a threat to external validity.

5. RELATED WORK

In this section, we describe studies that analyze bias in software engineering and bug localization studies. Our survey here is by no means complete.

Bias in Software Engineering: The most closely related work to ours is the work by Herzig et al. [5]. In that study, they investigate if issue report misclassification affects bug prediction. In this work, we extend their study to investigate if issue report misclassification affects bug localization. There are other studies that analyze bias in software engineering: Bird et al. note that only a fraction of bug fixing activities are labelled in version control systems and investigate the effect of this on the performance of bug prediction techniques [3]. Kim et al. investigate the level of noise that can affect the performance of defect prediction and propose an approach that can detect and eliminate noise [6].

Bug Localization: Lukins et al. propose the usage of a popular topic modeling algorithm, named Latent Dirichlet Allocation (LDA), to rank source files [7]. Rao and Kak evaluate the performance of a number of standard IR methods for bug localization and demonstrate that simpler approaches like vector space modeling and smoothed unigram model perform the best [10]. Zhou et al. propose a bug localization approach which uses an extended vector space model, named rVSM, that considers the size of source code files [13]. Wang and Lo propose a new bug localization approach that considers version history, similar report and bug report structure that outperforms many other approaches [11].

6. CONCLUSION AND FUTURE WORK

Herzig et al. have shown that more than 40% of the issue reports are wrongly classified. In this work, we investigate if these misclassifications affect bug localization. We compare the effectiveness of a bug localization tool on issue reports marked as bugs by their reporters (`Reported`) and issue reports that are actual bug reports (`Actual`). Comparing the results for `Reported` and `Actual`, our empirical study finds that there are -2.33%, 12.25%, and 6.98% differences in the mean average precision scores for HTTPClient, Jackrabbit, and Lucene-Java, respectively. Using Mann-Whitney U test, we find that the differences are significant. We have also studied the misclassification types that have the most impact to bug localization and make some recommendations to mitigate this impact.

In the future, we intend to investigate the impact of misclassification on other bug localization techniques, and the relative performance of these techniques on a cleaned dataset.

7. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Coping with an open bug repository. In *ETX*, 2005.

[2] A. Bachmann, C. Bird, F. Rahman, P. T. Devanbu, and A. Bernstein. The missing links: bugs and bug-fix commits. In *FSE*, 2010.

[3] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. T. Devanbu. Fair and balanced?: bias in bug-fix datasets. In *ESEC/FSE*, 2009.

[4] T. F. Bissyandé, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Réveillère. Empirical evaluation of bug linking. In *CSMR*, 2013.

[5] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *ICSE*, 2013.

[6] S. Kim, H. Zhang, R. Wu, and L. Gong. Dealing with noise in defect prediction. In *ICSE*, 2011.

[7] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. *Information & Software Technology*, 52(9):972–990, 2010.

[8] H. Mann and D. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[9] C. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge, 2008.

[10] S. Rao and A. C. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *MSR*, 2011.

[11] S. Wang and D. Lo. Version history, similar report, and structure: Putting them together for improved bug localization. In *ICPC*, 2014.

[12] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung. Relink: recovering links between bugs and changes. In *FSE*, 2011.

[13] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *ICSE*, 2012.