Int. J. Advance Soft Compu. Appl, Vol. 12, No. 2, July 2020
ISSN 2074-8523;
www.i-csrs.org

Improving Bug Localization using IR-based Textual Similarity and Vectorization Scoring Framework

Ginika Mahajan and Neha Chaudhary

Manipal University Jaipur
e-mail: ginika.mahajan@jaipur.manipal.edu
Manipal University Jaipur
e-mail: chaudhary.neha@jaipur.manipal.edu

Abstract

The major challenge faced by software industry is meeting deadlines in delivering quality product. The major reason behind delays is not only development part but basically detection and finding of bug or error. Whenever a bug is reported, developers use bug reports to reach to the code fragments that need to be modified to fix the bug. Suitable semantic information is present in bug reports and developers start exhaustive searching manually to catch the bug location. To minimize this manual effort, a framework on Information retrieval based bug localization is proposed that exploits the textual content of bug report to provide the rank relevant buggy source files i.e. the file having higher probability of occurrence of bug. The dataset used consists of a total of 925 bugs from 4 project categories SWT, ZXing, Eclipse and AspectJ. This framework outputs the Top N, here top (related) terms top 5 ranked sequence terms, showing the file containing these terms having higher probability of occurrence of bug.

Keywords: *Bug Localization, Bug Report, Information Retrieval, LDA, Vectorization Scoring Model*

1 Introduction

In software industry, bug localization has become a necessary activity to deliver projects with quality on time. Bug localization is a substantial task during various phases of software cycle in software testing, maintenance and quality assurance. Locating bugs is significant, challenging, and expensive, particularly for large-scale systems. Many times developers are not able to locate the root of bugs and hence lot of time and effort is wasted in finding the bugs manually. Hence bug localization has become an essential activity in software industry as to automate the process of finding the bugs.

Automatic localization of buggy files can speed up the process of bug fixing to improve the efficiency and productivity of software quality assurance team [1]. To address this, information retrieval techniques are increasingly being used to suggest potential faulty source files using bug reports. Researchers are working on numerous techniques and approaches of bug localization. Unfortunately, none of the technique contributes to 100% accuracy. But getting nearby location of bug helps software team to find the bugs with less effort and time.

Bug Localization (BL) process has two approaches- Information Retrieval-based BL and Spectrum- based BL. The basic variance is in the kind of input these approaches use, one is using bug reports and other program spectrum. The program entities that are intensely related with failures are identified as "suspicious", so that developers can examine them to see if they are faulty. The other way is to use bug reports that contains description about the bugs encountered. Figure 1 shows the bug localization overview where input can be bug report or source code entity. The output is the ranked list of program elements that are likely to contain bug.

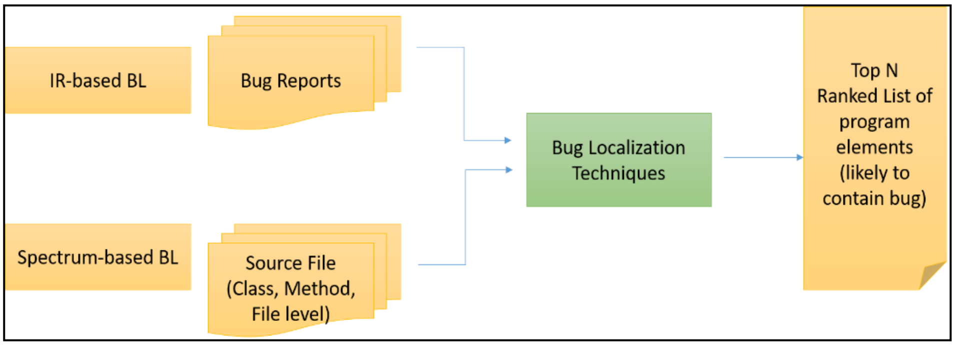

Fig. 1. Bug Localization Overview

2 Related Work

In recent years, researchers are working on various bug localization approaches using various techniques. The survey here is by no means complete. Comparison has been made on the basis of techniques, results and datasets used which is shown in Table 1.

A. Lam et al. [2] addressed a new approach DNNLOC which works on deep neural network (DNN) and rVSM IR technique. They used rVSM to collect the features based on text relationship. In this approach, DNN is used to match the terms in bug reports to different code tokens and terms in files. They found that by using these two approaches together they are able to achieve higher bug localization accuracy.

R. Gharibi et. al. [3] proposed a multi-component bug localization approach that works on various text properties of bug reports and source files. Also they are able to get relation between previously fixed bug report and a newly received bug report. They worked with text matching, stack trace analysis, and multi-label classification to improve the performance. It shows improvement in ranked list, MRR and MAP values compared to several existing bug localization approaches.

A. Kukkar and R. Mohana [4] proposed a hybrid approach wherein they merged the domains of text mining, NLP and ML to identify bug report as bug or non-bug. In their work, they used TF-IDF and Bigram methods to extract features and give classification results using K-nearest neighbor classifier. They worked on five different datasets of bug reports and evaluated accuracy based on Precision, Recall and F-measure values by using five datasets. Also its observed that using bigram method improves the performance of KNN classifier.

Yu Zhou et. al [5] proposed an approach that works in three stages where in the first stage summary part of bug report is used in Multinomial Naive Bayes Classifier. In the next stage these, now structured features, are used for prediction that can then be analyzed using Bayesian Net Classifier. And in last stage data grafting is done two bridge the two stages. Comparative experiments show enhancement (from 77.4% to 81.7%, 73.9% to 80.2% and 87.4% to 93.7%, respectively) in terms of overall performance.

T. Dao et al. [6] in their empirical study investigated dynamic execution information such as coverage, slicing, and spectrum information that can help with IR-based bug localization. They have cleansed the ranked list of suspicious localities produced by IR-based technique. They compared their results with previous baseline technique, BugLocator, and BLUiR and got better results.

K.Youm et al. [7] designed a combined method to integrate all the analyzed data to enhance the bug localization accuracy. BLIA is a statically integrated analysis approach of IR-based bug localization where it used texts and stack traces in bug reports, structured information of source files, and code change histories. Results shows that BLIA gave better results in terms of mean average precision when compared with existing tools BugLocator, BLUiR, BRTracer and AmaLgam.

Xin Ye and Chang Liu [8] introduced an adaptive ranking approach that worked on various parameters including bug fixing history, code change, dependency graph, API descriptions and functional decomposition of the program code. This approach also considered before fix version of bug report for better analysis. The authors used Learning to rank approach whose results proved that it outperforms other methods of bug localization.

R. Saha et.al [9] worked on C programs to find the effectiveness of IR based Bug localization on projects other than object oriented programming. In this paper they have created a dataset consisting of around 7500 bug reports from five popular C projects. The results showed that IR-based bug localization in C at the file level is overall as effective as in object oriented projects.

S. Thomas et al. [10] introduced a framework that combines the results of multiple classifier configurations as classifier combinations has shown promising results in other software domains. Also this paper empirically investigated around 3172 large space classifier and showed that the parameters of a classifier and combination of multiple classifiers improves the performance.

Table 1 Comparison of dataset and techniques used in Bug Localization

Reference	Dataset	Technique
[2]	AspectJ Birt Eclipse UI JDT SWT Tomcat	DNN and rVSM
[3]	AspectJ SWT ZXing	Information Retrieval, Textual matching, Stack trace analysis, and Multi-label classification
[4]	Mozilla Eclipse JBoss Firefox OpenFOAM	TF-IDF, Bigram and K-nearest neighbor (K-NN) classifier
[5]	Mozilla Eclipse JBoss Firefox OpenFOAM	Multinomial Naive Bayes Classifier and Bayesian Net Classifier
[6]	AspectJ Ant Lucene Rhino	Dynamic execution information- coverage information, slicing information, and spectrum information.
[7]	AspectJ SWT ZXing	Texts and stack traces in bug reports, structured information of source files, and source code change histories
[8]	Eclipse JDT9 Birt10 SWT11 Tomcat12 AspectJ13	Learning to Rank
[9]	C projects Python 3.4.0 GDB 7.7 WineHQ 1.6.2 GCC 4.9.0 Linux Ker	Adapted BLUiR for C code
[10]	Eclipse JDT IBM Jazz Mozilla mailnews	Multiple Classifier Configurations

3 Background

Information Retrieval-based Bug Localization

Developers commonly receive bug reports in huge number and debugging these reports manually is a challenging task that consumes much resources. Information Retrieval is a system of tracking and recovering specific information from stored data. It is an activity of obtaining information system resources relevant to an information needed from a collection [12].

IR-based bug localization assists developers in locating buggy source code entities (e.g., files and methods) based on the content of a bug report. IR-based bug localization techniques use query and document to get the relevance of document with query. Here query is taken as bug report and document as program elements. Figure 2 diagrammatically explains the basic overview of IR based BL wherein bug reports are taken as input and output program entities. The perception behind using these techniques is that program entities share various common terms with bug report and hence are possibly be relevant to the bug. Then, Accordingly the program elements are then ranked and sent to developers [3]. Developers then manually inspect output to locate source code segments that should be modified in order to fix the bug. Figure 3 shows a sample bug report used in this paper.

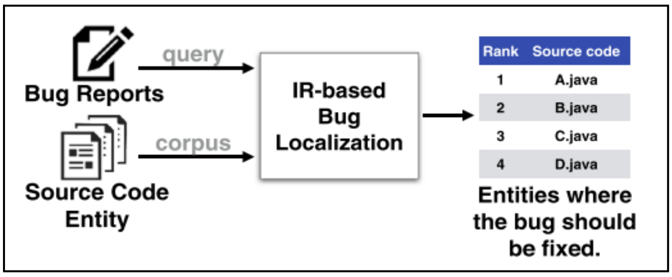

Fig. 2. IR-based Bug Localization [4]

```
<bugrepository name="SWT">
  <bug id="75739" opendate="2004-10-06 17:02:00" fixdate="2004-10-18 17:40:00">
    <buginformation>
      <summary>Variant has no toString()</summary>
      <description>The Variant class has no toString() and one cannot call getS
    </buginformation>
    <fixedFiles>
      <file>org.eclipse.swt.ole.win32.Variant.java</file>
    </fixedFiles>
  </bug>
```

Fig. 3. A sample bug report

4 The Proposed Method

With an objective to reduce developer's efforts and time in localization of bugs, this work proposes a novel Bug Localization framework based on Information Retrieval that uses Bug Reports as input and outputs a ranked sequence of terms of file names. This ranked sequence can be used by the developers to find the root cause file of the bug. Thus, this bug resolution activity will require considerably less time and effort to reach the bug and hence will be useful in improving the software quality and ensures its integrity.

This framework converts text data to features and features to vectors. We have implemented two models for feature representation, and a topic model that have been used in the field of information retrieval (IR). He framework is shown in figure 4 and its steps are explained in experimentation section.

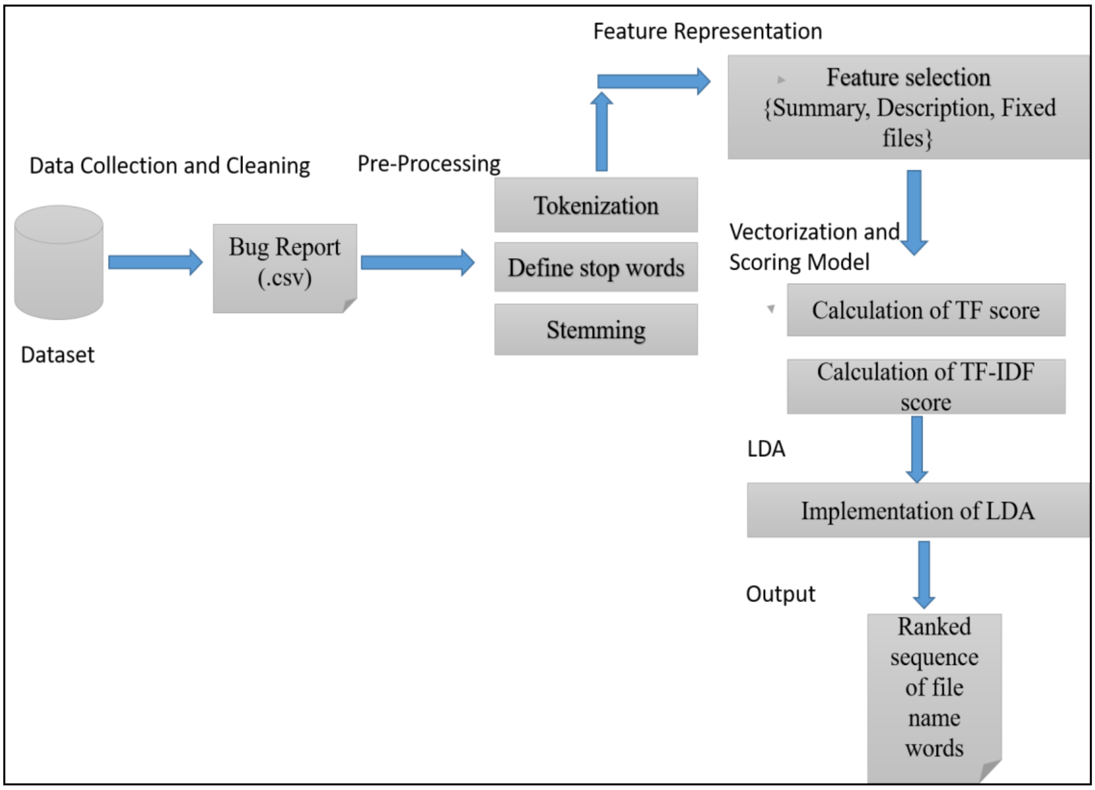

Fig. 4. Framework of proposed work

5 Results, Analysis and Discussions

The dataset consists of a total of 925 bugs from 4 project categories: SWT, ZXing, Eclipse and AspectJ shown in Table 2. Sample bug report is shown in Fig 3 which contains information of bug in the form of bug Id, opendate, fix date, summary, description and fixed file.

Table 2 Dataset used

Project	Number of Bugs
Eclipse	356
SWT	198
AspectJ	287
ZXing	84

The following steps are integrated and implemented to automate the proposed framework of Bug Localization:

Step 1: Data Collection

The dataset collected was in .xml format. For processing the bug report, this framework requires data in .csv format. Hence dataset is converted to .csv format and only required features are taken rest are removed. Figure 5 shows the converted and cleaned dataset with features bugId, fixed file, summary and description.

_id	fixedFiles/file/1	summary	description
78548	org.eclipse.swt.widgets.Too	Variant has no toString()	The Variant class has no toString() and one cannot call getString(
78854	org.eclipse.swt.dnd.DragSou	NullPointerException in CLabel.findMnem	I200411041200, GTK+ 2.4.9, KDE 3.3.0, Linux 2.6.9 I was creating ne
83262	org.eclipse.swt.dnd.TextTra	[consistency] Button Selection fires befor	- run the ControlExample, Button tab - turn on listeners MouseU
80830	org.eclipse.swt.graphics.GC.	[consistency] Slider fires two Selection ev	- run the ControlExample, Slider/Scale tab - turn on the MouseD
84557	org.eclipse.swt.widgets.Tre	[consistency] setItems(String[]) with null	Check if all platforms stop at null or ignore null elements.
87855	org.eclipse.swt.widgets.Tre	Sash no longer draggable when too small	Hi, see this snippet: public class Main { public static void main(St
87997	org.eclipse.swt.widgets.Tre	CTabFolder layout puts top right item one	see the attachment, I have a toolbar on top right and it cuts a pix
92017	org.eclipse.swt.graphics.Ima	CBannerLayout calls Control.update too of	CBannerLayout.layout() calls Control.update() all the time. Using

Fig. 5. Cleaned dataset in .CSV format

Step 2: Pre-processing

Pre-processing is the next phase to process and clean the input data in required form. Tokenization is performed to obtain groups of words which is followed by removal of all common separators, operators, punctuations and non-printable characters. Further filtering of stopwords that aims to get the most frequent terms is performed. Finally, stemming is applied to obtain the main words.

Step 3: Feature Representation and BoW Model

Bag of Words (BoW) model in IR represents text according to occurrence of terms in a file. If a term occurs in the document, then its value becomes non-zero in the vector and count increases as per its frequency of occurrence. We have applied CountVectorizer that converts a collection of text corpus to a matrix of term counts.

Step 4: Vectorization and Scoring Model

In this phase vectorization is done and TF-IDF are calculated. TF refers to Term Frequency and IDF refers to Inverse Document Frequency. Scoring Model uses these two metrics in its computation. Mathematical equations of TF x IDF is as follows [11]:

TF x IDF score for term “i” in document “j” = TF(i, j) * IDF(i)
TF(i, j) = (Term i frequency in document) / (Total terms in document)
IDF(i) = log2(Total documents / documents with term i)

For vectorization of of TF-IDF features, we have used TfidfVectorizer.

	_id	fixedFiles/file/1	Cleaned fixedFiles/file/1
0	78548	org.eclipse.swt.widgets.ToolItem.java	b'org eclips swt widget toolitem java'
1	78854	org.eclipse.swt.dnd.DragSource.java	b'org eclips swt dnd dragsourc java'
2	83262	org.eclipse.swt.dnd.TextTransfer.java	b'org eclips swt dnd texttransf java'
3	80830	org.eclipse.swt.graphics.GC.java	b'org eclips swt graphic java'
4	84557	org.eclipse.swt.widgets.TreeColumn.java	b'org eclips swt widget treecolumn java'

Fig. 6. Feature Representation

Step 5: Latent Dirichlet Allocation

LDA is a statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar [4]. Here Topics are represented as a collection of terms. They are very valuable to summarize large corpus of text documents and further they reveal latent patterns in the data. Here we get four topics as shown in Fig 7 containing the Top N, here top (related) terms top 5 ranked sequence terms, showing the file containing these terms has higher probability of bug and hence is considered as root cause or most probable location of respective bug.

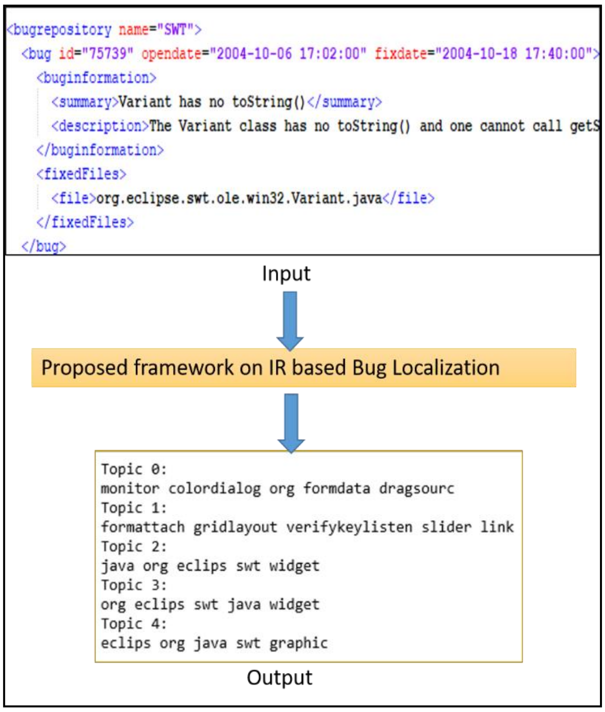

Fig. 7. Input and output representation of framework

6 Conclusion and Future Work

The key challenge software industry is facing is of often shipping the product with defects and not meeting the deadlines. Fixing the defect or bug is not a big issue but the major time is consumed is reaching and locating the root of bug. To minimize the manual effort, this framework is proposed that exploits the textual content of bug report to provide the rank relevant buggy source files i.e. the file having higher probability of bug. This work can be extended in getting more precise file paths and further applying learning to rank using RankLib tool to give better ranking results.

References

[1] Saha, R. K., Lease, M., Khurshid, S., & Perry, D. E. (2013, November). Improving bug localization using structured information retrieval. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 345-355). IEEE.
[2] Lam, A. N., Nguyen, A. T., Nguyen, H. A., & Nguyen, T. N. (2017, May). Bug localization with combination of deep learning and information retrieval. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*(pp. 218-229). IEEE.
[3] Gharibi, R., Rasekh, A. H., Sadreddini, M. H., & Fakhrahmad, S. M. (2018). Leveraging textual properties of bug reports to localize relevant source files. *Information Processing & Management*, *54*(6), 1058-1076.

[4] Kukkar, A., & Mohana, R. (2018). A Supervised Bug Report Classification with Incorporate and Textual field Knowledge. *Procedia computer science*, *132*, 352-361.
[5] Zhou, Y., Tong, Y., Gu, R., & Gall, H. (2016). Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, *28*(3), 150-176.
[6] Dao, T., Zhang, L., & Meng, N. (2017, May). How does execution information help with information-retrieval based bug localization?. In *Proceedings of the 25th International Conference on Program Comprehension* (pp. 241-250). IEEE Press.
[7] Y. oum, K. C., Ahn, J., & Lee, E. (2017). Improved bug localization based on code change histories and bug reports. *Information and Software Technology*, *82*, 177-192
[8] Ye, X., Bunescu, R., & Liu, C. (2016). Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation. *IEEE Transactions on Software Engineering*, *42*(4), 379–402.
[9] Saha, R. K., Lawall, J., Khurshid, S., & Perry, D. E. (2014, September). On the effectiveness of information retrieval based bug localization for c programs. In *2014 IEEE International Conference on Software Maintenance and Evolution* (pp. 161-170). IEEE.
[10] Xia, X., Lo, D., Wang, X., Zhang, C., & Wang, X. (2014, June). Cross-language bug localization. In *Proceedings of the 22nd International Conference on Program Comprehension* (pp. 275-278). ACM.
[11] Khatiwada, S., Tushev, M., & Mahmoud, A. (2018). Just enough semantics: An information theoretic approach for IR-based software bug localization. Information and Software Technology, 93, 45–57.

Notes on contributors

Ginika Mahajan is Assistant Professor at the Department of Information Technology, Manipal University Jaipur, India. Her main teaching and research interests include Software Testing, Machine Learning and Text Mining. She has published several research articles in international journals of software engineering.

Dr. Neha Chaudhary is Associate Professor at the Department of Computer Science and Engineering, Manipal University Jaipur, India. Her main teaching and research interests include Software Testing, Quality Assurance, Data Mining and Data warehousing. She has published several research articles in international journals of software engineering.