BUG LOCALIZATION AUTOMATION USING COMBINATION OF IR TECHNIQUES

Mr. S. Mastan [1]
3rd Year Student,
Department of Computer Science,
SV U CM & CS, Tirupati.

Dr. E. Kesavulu Reddy[2],
Assistant Professor,
Department of Computer Science,
SV U CM & CS,, Tirupati.

ABSTRACT: Bug localization is the technique to locate the relevant source code from bug reports in order to fix that issue. Usually developers do it manually, hence take lot of time and efforts which leads to high maintenance cost. Our effort is to reduce that cost by applying the more efficient Information Retrieval techniques for bug localization. We are going to use the combination of various methods in order to reduce the efforts and time for bug localization and also to provide the automated framework. This framework consists of major four phases as bug and source code preprocessing as removal of stack traces and unnecessary data, then applying concept location with page rank algorithm to find the relevant source code for the given bug also further applying Relational Topic Model for finding the relevant source code for the given bug. Then combine these two techniques and by Reciprocal Rank Fusion method compare its results and get the best relevant source code files.

I INTRODUCTION

Bug Localization can be defined as an information retrieval problem. It can be mentioned as a classification problem as given m source code entities and a bug report r, now classify the bug report r as entity belonging to one of the source code entities m. This way by classification the relevant source code files will be found relevant to the bug. Bug localization is the task of finding the relevant source code entities which needs to be modify in effort to fix the bug. Most of the time in industry this task is done manually. This task is time consuming and effort consuming as well. Still most of the industry follows the manual bug localization method. Once it has been realized that bug localization is a classification problem and can be solved by using Information Retrieval techniques, it leads researcher to automate it by using traditional Information Retrieval techniques. Once this problem is automated the implications are that it will require less manual efforts and less time. Hence it will lead to less maintenance cost of the software. So automation for bug localization can be a major factor in attempts to reduce the software maintenance cost. It uses the basic fact that classifier configuration makes a significant impact so we need to choose the parameters carefully and the use of combination of classifiers give better results [1].

II LITERATURE SURVEY

Information Retrieval is the study of querying for text within a collection of documents [2]. It is more or less finding some entity through the search engine. Hence, the IR techniques play a major role in bug localization. Much research is done on various IR techniques used for bug localization. The research of Rao and Kak employed several popular IR techniques for bug localization and evaluated their performances [3]. Rao and Kak's work includes evaluating the various IR models as VSM (Vector Space Model), LSI and LDA and various combinations. They performed a case study and concluded that the simpler IR models often outperform more sophisticated models. Lukis et.al.Applied Latent Dirichlet Allocation (LDA) for bug localization [4]. Using LSI and LDA he build the two classifiers on the identifiers and comments of the source code and compute the similarity between a bug report and each source code entity using the cosine and conditional probability similarity metrics. His conclusions were based on performing the experiments on Eclipse and Mozilla bug reports and concluded that LDA often outperforms LSI. Neguyen et al. [5] worked on a new Topic Model which was based on the earlier IR model LDA, it was called BugScout. It mainly considered the past bug reports in addition to the identifiers and comments.

Along with IR techniques there is tool which can be plug-in in bug tracking system and version control system and helps in performing bug localization online. Such tool is bug localizer [7]. It is based on Zhou et al [8]. It is implemented as Bugzilla extension, it extracts information from summary and description parts and uses revised VSM and bug file graph from past similar bug reports. So based on the past source code entities which developers changed at that time, developers can get links for this similar bug.

III PROPOSED WORK

The suggested framework is divided into four major modules. The intention of this system is to use the combination of two IR techniques as Concept Location and Relational Topic Model. First on source code query perform concept location and get its results and then on the same source code and query apply Relational Topic Model. After that the system will use the Reciprocal Rank Fusion method and take out the best results from Concept Location and Relational Topic Model and will deliver the most relevant source file which tops on both models as output of the suggested framework.

The first module is Bug and Source code Preprocessing. It uses previously used techniques combined in one module. The bug report and source code is considered to be the input and these two entities are preprocessed to remove the noise from the bug and source code.

The second module is the 'Concept Location'. Here we use the concept location technique with Page Rank algorithm for bug localization.

The third module is Relational Topic Model. Like Concept Location module, this module will also find the relational topics based on the link importance and link structure which are the relevant source code files and rank them as per their relevance.

The forth module is the module for combinational techniques used for combining the IR classifiers/techniques. In this system the used combinational framework is Reciprocal Rank fusion which combines studies of Concept location and Relational Topic Model and finds the best relevant source code.

A software bug is a error, flaw, failure or fault in a computer program or system because of which the intended program, system is not meeting the desired results as expected. To achieve high quality software engineering tasks have included software testing tasks to start side by side with development activities. When the initial software is ready to test then that version goes to software testers who test those scenarios as per the customer requirements/system requirements. Testing is the conformance to the requirements. Testers test various scenarios and log the defect/flaw/bug in some defect tracking tool so that later developers can check it and find the source code which is the root cause for such error and make necessary changes to the source code files and fix the defect. The Defect life cycle starts when the defect is found by the tester and he/she logs it in the defect tracking system. The different states defect goes through its life cycle are as below;

1. New: When a defect is logged for the first time by the tester.

2. Assigned: After defect is logged by the tester the test lead verifies and approves the defect as genuine defect and assign the bug to the corresponding developer or developer team.

3. Open: Here in this state the developer starts analyzing and working on the defect. 4. Fixed: When developer makes necessary changes to the source code files to remove the error/bug, he changes the state as 'Fixed'.

5. Retest: At this state the tester again tests the functionality/bug and verifies that whether the changes made by developer are adequate and functionality is working as expected.

6. Verified: Once the tester has tested and confirmed that the functionality is working as expected then he/she changes the state as 'Verified'. It is the assurance that what the developer has changed in source code that has been effective and without creating any further error the error has been removed.

7. Reopen: While testing the bug fix if the tester feels that the issue is not fixed and error still persists then he/she changes the state as 'Reopen' and then then the developer should work again on that and the bug follows the whole cycle again.

8. Closed: Once the tester is assured about the bug fix then he/she closes the bug and changes its state as 'Closed'.

9. Duplicate: Many testers are working simultaneously so there is possibility that same bug is logged by others. In such cases only one copy is kept and others are marked as 'Duplicate' and will not be entertained. 10. Rejected: In many scenarios the development team might be in disagreement of a bug in such scenarios with consultation and approval with customer/client/analysts/end stakeholders development team marks the bug as 'Rejected'.

11.Deferred: In many situations the based on the priority and timeline and severity few bugs are 'Deferred' to be fixed in later releases.

IV IMPLEMENTATION DETAILS AND RESULTS

GROUND TRUTH AND BASE DATA

To compare the results of this framework first some ground truth needs to be in place. To compare later we are going to use some already registered bugs and their manually found source files by developers. This is the manual bug localization. Later on the same code and bug and on the same version of the code we will implement our framework and compare the results in terms of accuracy and time.

In this section the basic reported bug and its associated manual source file is considered. On the very same data we

will be implementing our code and later in this section the results are mentioned.

EXPERIMENTAL RESULTS

To get the idea of first module Bug and Source code preprocessing the system will take an example of a bug mentioned below as Bug B. Once the bug B enters into the system as input first module preprocesses Bug and preprocesses the source code based on the parameters mentioned.

Bug Details- We are using here eclipse JDT bug report and the source code dataset is Eclipse JDT.

PERFORMANCE METRIC

To check the actual performance and accuracy of this suggested framework it should be based on the manual bug localization; the file relevant file manually found by the developer and the relevant file found by our suggested system. We need to compare these two results and if they match then it means this suggested system works accurately.

To measure the performance of used classifiers we are using top-k accuracy metric. Many renowned researchers have used the same method so this paper will also follow the same method for measuring the performance of used classifier combinations.

V CONCLUSIONS

In this paper we have seen that there are various IR techniques used for bug localization, even some combinations of IR classifiers are used for the same. They give better results than individual classifiers still there is a need to do further research in other combinations of classifiers/concept location which exactly is our area of research in this paper. Here we have suggested a combinational framework which consists of preprocessing of bug and source code, implementing concept location for finding relevant source files as a query result for given bug, the same we are trying to apply on Relational Topic Model and with Reciprocal Rank fusion/Score addition we will combine the results and will find which is the best candidate relevant to the files bug and that will be the bug localization result of this system. In future we intent to show that how it is less time consuming and equally accurate with manual bug localization. Such automated combinational frameworks definitely will reduce the effort and time on developer's side and hence will reduce the software maintenance cost.

VI REFERENCES

[1] S. Thomas, M.Nagappan, D.Blostein,A.Hassan, "The Impact of Classifier Configuration and Classifier Combination on Bug Localization", IEEE Transactions on Software Engineering, vol. 39no. 10, pp. 1-2,2013.

[2] C.D. Manning, P. Raghavan, and H. Schutze, Introduction to Information Retrieval, vol. 1, Cambridge Univ. Press Cambridge, 2008

[3] S. Rao and A. Kak, "Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models," Proc. Eighth Working Conf. Mining Software Repositories, pp. 43-52, 2011

[4] S.K. Lukins, N.A. Kraft, and L.H. Etzkorn, "Bug Localization Using Latent Dirichlet Allocation," Information and Software Technology, vol. 52, no. 9, pp. 972-990, 2010

[5] A.T. Nguyen, T.T. Nguyen, J. Al-Kofahi, H.V. Nguyen, and T.N.Nguyen, "A Topic-Based Approach for Narrowing the Search Space of Buggy Files from a Bug Report," Proc. 26th Int'l Conf. Automated Software Eng., pp. 263-272, 2011

[6] Automation for Bug Localization using combination of IR techniques,Ms. Dhanashree P. Pathak,Computer Engineering Department,G.S.M.C.O.E., Balewadi,Savitribai Phule Pune University, Pune, India,

[7] R.W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems," IEEE Trans. Software Eng., vol. 31, no. 6 pp. 495-510, June 2005.

[8] C.D. Manning, P. Raghavan, and H. Schutze, Introduction to Information Retrieval, vol. 1, Cambridge Univ. Press Cambridge, 2008

[9] G. Salton, A. Wong, and C.S. Yang, "A Vector Space Model for Automatic Indexing," Comm. ACM, vol. 18, no. 11, pp. 613-620,1975. [9] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R.Harshman, "Indexing by Latent Semantic Analysis," J. Am. Soc.Information Science, vol. 41, no. 6, pp. 391-407, 1990.