International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Survey Paper for Bug Localization

Dhanashree P. Pathak[1], Srinu Dharavath[2]

[1, 2] Savitribai Phule Pune University, G.S.Moze College of Engineering, Balewadi, Pune-411045, India

Abstract: ***Bug localization is the task to locate the source code entities which are relevant from the bug report. Manual bug localization is a time consuming and labor consuming task as developers has to go through thousands of source code entities to locate the relevant one. Current research provides methods as various IR techniques, classifiers, combination of classifiers to improve bug localization. To make bug localization partial or maximum automated some tools are available but mostly they are based on simple query, IR techniques or eliminating unnecessary stack traces or use basis of previous bug reports and its changed files and based on that history relevant source files will be identified. There is automated path generation for software fault localization. But none of all these tools/techniques achieves highest efficiency as they work only on one area or combination of areas. To make automation more successful and efficient there is a need for finding hybrid approaches. This paper provides the literature survey about what is done regarding bug localization so far and what is the future scope for automation in bug localization. It also points how efficient automated bug localization may help maintain the software cost.***

Keywords: Bug Localization, Software maintenance, Information Retrieval, Query Expansion, Classifiers, Automation of Bug Localization.

1. Introduction

This survey paper has been divided into four sections. Section 1describes what is bug, bug life cycle and describes one can maintain software cost if we speed up bug fixation process and also states 'bug localization'. Section 2 points bug localization as IR(Information Retrieval) problem, various methods used so far and discuss its effectiveness. Comparison between IR methods, classifier methods, some research work. Section 3discusses the shortcomings of above techniques to achieve maximum efficiency and need for proper automation. Discussion about what automation tools/techniques and their basis have been researched so far. Section 4 suggests our proposed research work for automation of bug localization as to use the hybrid approach for combining different right classifier combination and combination of classifier combinations to achieve more efficiency and successfully automating it in addition with concept location with Relational Topic Model/BugScout/BM25F, Query expansion, Preprocessing bug reports by removing noise from stack traces and code snippets. By combining all, creating a hybrid automated model which may lead to better efficiency.

1.1 Bug, Bug Life Cycle, Bug Localization

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.

From the above bug cycle we want to concentrate on the states 'OPEN' till 'FIXED' as these are the states involved in bug localization. Now once the defect is 'OPEN' and developer is working on that he/she should go thoroughly through the bug report and should try to find out the relevant source code entities which has caused the error in order to fix the bug. This process is called as 'Bug Localization'[1][2][3]. Considering the source code is a large entity and through bug report finding relevance to that is a considerable time and effort consuming task. Often it has been seen that the there are large number of source files while the bug usually affect only a few number of files. Lucia et.al reported that 84-93% of bugs only affect 1-2 source code files[4]. As the large number of bug reports can overwhelm the developers, for instance, in the Eclipse project, developers receive an average of 115 new bug reports every day, the Mozilla and IBM Jazz projects get 152 and 105 new reports per day, respectively[5]. The current bug localization efforts are manual which increases the fixing time as to manually locate appropriate entities and which is difficult[6] and expensive[7]. As the time for bug fixation increases the overall time for software development increases and the total software cost increases in proportion. So to maintain the software cost we must control the bug fixation time to minimal. Here comes the need of using the effective IR techniques, different classifier approaches and also automating the bug localization.

2. Information Retrieval Models For Bug Localization

Information Retrieval is the study of querying for text within a collection of documents [8]. It is more or less similar as finding some keyword from Google engine. In the similar way IR based bug localization classifiers use IR models to find textual similarities between the bug report(query) and the source code entities(documents). If a bug reports contains "Trimmed 30 bytes off each pageRequest object", then an IR model looks for the source code entities which contains the words "trim", "bytes", "pageRequest" etc.

Bug localization can be defined as a classification problem as : Given the n source code entities and a bug report b, classify the bug report b as belonging to one of the n source code entities. The classifier returns the ranked list of possibly relevant entities, along with a relevancy score of each entity in the list. An entity is considered relevant if it indeed needs to be modified to resolve the bug report and irrelevant otherwise[5]. Current bug localization techniques uses IR techniques. We will see some popular IR techniques and their comparisons for bug localization and related research work by others.

2.1 Vector Space Model (VDM)

The Vector Space model is a simple algebraic model based on the term-document matrix of a corpus[9]. The term-document matrix is a m × n matrix whose rows represent individual terms(i.e. words) and columns represent individual documents . The ith and jth entry in the matrix is the weight of term wi in document dj. The vector space model represents documents by their column vector in the term-document matrix , a vector containing the weights of the words present in the document and zeros if not. The similarity between the two documents is calculated by comparing their two vectors.
VSM uses the following parameters:

Term weighting (TW): The weight of a term in a document. It is like the number of occurrences of the term in the document. Or tf-idf i.e. term frequency, inverse document frequency [8].

Similarity metric (Sim): The similarity between two document vectors. It is Euclidean distance, cosine distance, hellinger distance, KL divergence.

2.2 Latent Symantic Indexing

Latent Symantic Indexing is a extension to VSM. It uses Singular Value Decomposition (SVD)to project the original term-document matrix into three new metrics. These three new matrices are used as ; a topic document matrix D, a term topic matrix T and a diagonal matrix S of eigenvalues [10]. Here the terms which are related by collocation are grouped together into "concepts" or "topics". For example in any computer related document the words "monitor", "keyboard"," mouse", "printer" are tend to appear in the same document as they are related to the same subject/topic. This reduced dimensionality of topic-document matrix has increased the performance over the VSM in some areas. LSI vectors contains the weights of topics whereas VSM contains the weights of single terms. LSI and VSM can use the same similarity measures to determine the similarity between the two documents. Here Term weight (TW) and Similarity Metric(Sim) are same as VSM. Only Number of Topics (K) is the parameter which controls how many topics are kept during the SVD reduction.

2.3 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation [11] is a statistical topical model which provides automatically index, search and cluster documents that are unstructured and unlabeled [12]. LDA discovers the "topics" from documents as first task same as LSI. The key difference between LSI and LDA is the method used to generate the topics. In LSI the topics were generated as byproduct of the SVD reduction of the term-document matrix. But in LDA topics are created through a generative process using machine learning algorithms.
LDA uses following parameters:
Number of topics (K) : IT decides how many topics to be created.
α : A document- topic smoothing parameter
β : A word-topic smoothing parameter.
Similarity metric (sim) : Similar as VSM sim.

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[1]. 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.[2] 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. When finding the key search concepts it used both data sources concurrently and concluded that BugScout improves the performance by 20% over the sole use of LDA only to source code.

Along with these IR based techniques there are also many data processing techniques are used to work out the bug localization. Preprocessing the bug reports always improves the results. Removing unnecessary data from data set and then searching for required data makes more sense. The basic stopping, stemming and splitting activities are done first. To move forward only to keep the necessary data many techniques includes the query expansion for searching in source code entities based on the bug report entity. Also the redundant bug reports gets removed as many times, once the developers identify the relevant entity using bug localization they use the change propagation techniques [13] to identify the other entities that also be modified. Removing noises from stack traces and source code preprocessing is also been the area of research for improving the bug localization along with IR techniques.

Also for bug localization some IR-based concept/Feature location approaches can also be helpful. The common thing between both approaches is finding the relevant source code entity to the given query.

In above all VSM, LSI, LDA IR models use these IR classifiers to locate source code entities that are textually similar to bug reports. However the current results are ambiguous and contradictory as some claim VSM provides the best performance [3]and some claim LDA [1]. Some claim new IR model is required [2]. These mixed results are based on different data sets and mainly the different classifier combinations. A classifier combination defines the value of all the parameters that specify the behavior of a classifier, such as the way in which the source code is processed, how terms are weighted and mainly the similarity between the bug report and the source code entities. But given that the range of parameters is vast we simply cannot use all combinations as it is highly difficult to understand exactly which parameters to consider and which to left out. The work of Thomas, Nagappan, Boistein and Hassen [5] has delivered the limitations of current research and how researchers and practitioners are left to guess which configuration to use for their project. They have come up with the discoveries which might improve the performance based on classifiers combination and classifier configurations. After considering IR based classifiers and entity metric based classifiers and evaluating the results they have concluded that the configuration of IR based classifier matters. [5] The best IR based classifier uses VSM with index built using tf-idf term weighting on all available data in the source code entities which has been stopped, stemmed and split and queried with all available data in the bug report with cosine similarity [5]. Classifier combination helps in almost all cases, no matter the underlying classifiers used or the specific combination technique used [5]. They have proposed two frameworks, one for defining and analyzing the classifier configurations and one for combining the results of disparate classifiers. The configuration of a classifier has a significant impact on its performance.

3. Practical Better Ways for Bug Localization for Researchers

So far we have seen concept of bug localization and how it can be done with various IR techniques, how we can improve it with right classifier configurations and should combine them in most effective way to accelerate the time and efforts for finding and fixing the bugs and decreasing the maintenance cost. But there are some practical shortcomings to achieve these techniques in practical day today development cycle for at least small/mid size companies. First thing is the accessibility of the tools. It requires the developers to download the bug reports and source code files and run techniques to localize bugs. The better solution might be the 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 [14]. It is based on Zhou et al [15]. 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.

There is another tool available for researchers called as BOAT (A bug localization experimental platform) [16]. In this web application researchers can use their newly proposed bug localization techniques and compare them against the existing techniques. This tool is already loaded with thousands of bug reports and source code entities. Developers and managers can use this application to reduce their manual efforts for bug localization and hence decrease the maintenance cost. Still bug localization is in its preliminary stage as it has not reached the level where it is completely automated.

4. Future Scope for Bug localization to make it automated completely

As of now still in industry bug localization is mainly done manually. To use the full potential of bug localization the scenario should be where majority of bug localization has become automated. Now much research has been done on what IR techniques needs to be used for bug localization. In

this paper we have seen many IR based techniques, their comparisons also seen research for classifiers and their combinations. In section three we have seen some tools that can be used by researchers to propose new techniques. The future scope of this paper or conclusion we can say is there is still many areas of research needs to be experimented as addition of formal concept analysis[17], static analysis, Relational Topic model [18], concept location need to fully investigate many possible combination techniques but mainly as the previous research suggests about classifiers we should need to do research on hybrid techniques/combination techniques as using two ,three methods for bug localization as using combination of classifiers and along with that using the Relational Topic Model along with using preprocessing steps for bug reports and query expansion techniques and removing noise from stack traces and code snippets. There should be automated software that will have the combination of all above mentioned techniques. It will ensure the 2-3 layers/2-3 projections simultaneously for achieving bug localization and by then we can test and say it improves the efficiency and hence decreases the maintenance cost. This combination techniques and their feasibility together needs more specific research.

References

[1] 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
[2] 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.
[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] Lucia, F. Thug, D. Lo and L.Jaig, “Are faults localizable?”, MSR, pp 74-77,2012.
[5] 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
[6] R.L. Glass, Facts and Fallacies of Software Engineering. Addison-Wesley Professional, 2003
[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.
[10] 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.
[11] D.M. Blei, A.Y. Ng, and M.I. Jordan, “Latent Dirichlet Allocation,”J. Machine Learning Research, vol. 3, pp. 993-1022, 2003.
[12] M. Blei and J.D. Lafferty, “Topic Models,” Text Mining:Classification, Clustering, and Applications, pp. 71-94. Chapman &Hall, 2009.
[13] R. Arnold and S. Bohner, “Impact Analysis—Towards a Framework for Comparison,” Proc. Int’l Conf. Software Maintenance, pp. 292-301, 1993.
[14] Ferdian Thug, “Bug Localizer: Integrated tool support for bug localization”[,pp. 767-770, FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014.
[15] J. Zhou, H. Zhang, and D. Lo, "Where Should the Bugs
[16] Be Fixed?—More Accurate Information Retrieval-Based Bug Localization Based on Bug Reports," Proc. 34th Int'l Conf. Software Eng., pp. 14-24, June 2012.
[17] Xinyu Wang, "BOAT: An Experimental Platform for Researchers to comparatively and reproducibly evaluate bug localization techniques",ICSE Companion, pp. 572-575, 2014
[18] D. Poshyvanyk and A. Marcus, “Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code,” Proc. 15th Int’l Conf. Program Comprehension, pp. 37-48, 2007.
[19] J. Chang and D.M. Blei, “Relational Topic Models for Document Networks,” Proc. 12th Int’l Conf. Artificial Intelligence and Statistics,pp. 81-88, 2009.

Author Profile

Dhanashree Pathak received B.E. from Walchand Engineering College, Sangli in 2001. After graduation Dhanashree was associated with TKTE’s Engineering College in capacity of associate professor. From Year 2005 to 2013, she was handling various assignments with software testing, product services and was associated with reputed IT services companies in India as well as in US. Since June 2013 she is associated with G.S.Moze Engineering College as associate professor for computer engineering department.

Srinu Dharavath is B Tech in Computer from GEIT and M Tech in AI from Hyderabad University. He is currently associated with G.S. Moze Engineering college as Associate professor.