Review of Advancement in Bug Localization in Automated Software Engineering

Shashank Bipin Kumar
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA
sbkumar@ncsu.edu

ABSTRACT

Automated tools for accurate bug localization in field of software development can help reduce defect turnaround time significantly. This would allow developers to concentrate on developing quality products instead of defect fixing. This paper reviews few significant approaches proposed in field of bug localization between 2008 to 2015. This paper also examines the literature, suggesting necessary changes, scope of improvement and future work.

KEYWORDS

Defect Localization, Latent Schematic Indexing, Information Retrieval, Cost-sensitive classification, Search Length, Software Bug

1. INTRODUCTION

Frederick Brooks wrote that "Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level)" [1]. This inherent complexity in software construction makes defects in software frequent. When a bug is found in software system, typically a bug report is created where description of situation(s) when bug can be seen is reported. When a developer is assigned to solve a bug, he refers to the bug report to analyze the bug, search through the bulky software code to locate the potential buggy file. This process is called bug localization.

For a large scale software systems, the number of bugs reported might be hundreds or even thousands. Performing the bug localization process manually in large scale system development is a time consuming and costly process. This calls for developing techniques for accurate and automatic bug localization techniques which would help reduce bug turnaround time significantly. This would effectively result in a development of quality software product.

To best of my knowledge, the paper " Software Errors and Complexity: An Empirical Investigation" by Basili et.[2] al published in 1984 is a one of the first attempts to study the complexity of different software bugs, ways to categorize them and save on bug turnaround time. Subsequently there has been several study and proposals to solve this problem based on different automated models. Recently, information retrieval based bug localization models which have been proposed by software scientists. Authors claim that these models based on different data mining techniques have more accurate result.

Considering the vast development in field of automated bug localization, this paper would review the advancements in the domain, attempt to study how the issues faced by *previous papers been addressed or ignored or extended by subsequent papers.*

The motivation behind this exercise is mentioned in section 2, which is followed by Background in section 3, Related Work in Section 4, Survey Section 5,Dataset in Section 6 and Areas of Improvement in Section 7. The paper ends with Conclusion in Section 8 and References in Section 9.

2. MOTIVATION

To be successful software industry needs to build quality software, generally at a large scale. Building such software will always have inherent bugs. Solving maximum number of such bugs in a limited time period will help release quality software products on time and generate revenue.
Manually reproducing software issues, finding buggy piece of code in code repository often proves to be a cumbersome task. Creating automated self-learning models for bug localization can go a long way in reducing bug investigation time and hence allow developer's focus on development activity.

In [3], Nyugen et. al have published case studies where they have examined bug reports from 3-year development activity in a large corporations. They have observed that technical aspects shared by bug reports and source files can help locate the relevant source code for a particular bug. In [4] and [5] authors have observed that using history of bug investigation can prove to give accurate results for bug localization. These results and others discussed later in the paper gives ample motivation to study literature and understand techniques that could help in developing models for automatic bug localization.

3.BACKGROUND

Since few papers discussed in Survey section later would talk about Information Retrieval(IR) based bug localization, it would be apt to discuss about it before diving into the survey section.

The fundamental assumption underlying IR based bug localization is that some terminologies used in the bug reported would also be present in the source code files that would be needed to be fixed to solve the bug.

In IR based bug localization, source code files of the software represent a collection of document which would be search and the bug report is the search query. The better search result is then ranked using the standard IR ranking based on the relevance of used search query. An IR based system comprises of three step processing: text normalization, stop-word removal and stemming. Text Normalization consists of taking the code and bug, removing the punctuation, tokenizing, case folding etc to convert the raw text in form representative of document and query respectively. Stop-word removal comprises of reducing spurious and duplicate matches from the documents. Stemming would conflate variants of same underlying term which would improve term matching between query and document.

Once the queries and documents are pre-processed, documents are indexed with parameters like term frequency, which is number of times a term appears in a given document and document frequency, which is the number of document each term appears. These parameters would be then tuned and would help in ranking of pages which would *help in giving ranked results for a bug query and thus help in bug localization.*

4. RELATED WORK

Mockus and Weiss[10] studied 15000 maintenance requests which was raised in a period of ten years for a large telephone switching system to construct a model to predict the probability of software changes based on a request resulting in a defect in a software system. Their model took into account parameters like size of the change, number of sub-systems effected by change, duration needed to complete change, number of developers who were involved for making change. Their model used logistic regression and just predicts either there would be a failure by a change or not. However it does not predict the parts of the code where the fault could be found. Khoshgoftaar et al[9] have presented a model to create a binary decision tree to classify the modules based on fault prone or not.

Graves et al.[7] performed study to understand the characteristics of modules that had faults and developed models to predict the number of faults that would be associated with future version of the modules under study. They concluded that size of the module was poor predictor whereas the module's age, number of changes made and age of those changes were good predictor of fault likelihood.

Marcus et al.[11] have used a IR-based technique called latent semantic analysis (LSA) to document indexing and retrieval. Poshyvanyk et al. [12][13] have used latent semantic indexing (LSI) for information retrieval model. LSI is based on vector space model which is an algebraic model that represents text documents as vector of terms and represents the relationships between documents and terms. LSI uses singular value decomposition (SVD) to reduce co-occurrence matrix. Though this model was found to be more accurate than ones seen before but the results returned by LSI could be difficult to interpret considering it was represented using numeric spatial representation. Also LSI is bad when recognizing same term having multiple meanings. To solve this issue, Hofmann [14] tried to introduce the concept of probabilistic LSI(pLSI). Though the pLSI improves accuracy of search query over LSI significantly, it also adds a lot of terms which grows linearly and the model is left to read a lot more bulky processed data than that of LSI. This made the performance of the system to be worse than one using LSI.

Zhao et al. [17] have proposed BugLocator, which combines TEIDF formulation, a model heuristic for file length and knowledge of previously available bugs. The authors have made the test dataset and executable available in public domain. This has become benchmark for testing and comparing alternative IR based bug localization approaches.

5. SURVEY

This section would discuss the eight papers reviewed during course of the semester beginning with the paper published first i.e. in 2006 and moving sequentially towards paper published last. Each sub-section is titled same as the title of the paper reviewed.

5.1 Looking For Bugs in All the Right Places

This paper was published in 2006 by Bell et al.[3]. The work mainly constitutes investigation of use of a binomial regression model to predict files that are most likely to contain mainly faults in a large scale software system. The authors have used file characteristics that could be objectively accessed like size of the file based on the lines of code, the version of the software, changes in the file and history of number of bugs found in the bug.

In contrast to previous predictability model [18] based on Poisson's regression, negative binomial regression model was designed to handle outcomes that are non-negative integer like the number of faults in a file during a specified period. Poisson regression, which assumes that faults occur at random at a rate explained by a set of predictor variables. The model was also designed to aggregate the defect predicted based on the month and the year.

The authors have tested their model on automated voice response system's inventory resource and service provisioning resource. The authors have found that for both the systems the files that were ranked top 20% by their model contained 73 and 74 % of the total faults reported.

The problems with this model are that the authors have removed the earlier releases from their data during pre-processing and used regression test-suite to get the metric for number of faults reported in a file. Regression tests are run very late in test cycle and so the number of bugs reported for a file would not represent a realistic figure. Secondly when the software is developed, the earlier few release are the ones where a lot of bugs are found and pruning them from model data was a problem. The authors have also not used metric called new development feature. Typically a newly developed feature in software system are prone to have more bugs than rest of the code.

5.2 Extraction of Bug Localization Benchmarks from History

This paper was published by Dallmeier & Zimmermann in 2007. This contribution of this paper was creation of a tool to semi-automatically extract bug localization benchmarks from project history which could be later used to train bug localization tools. The authors have assembled all their data into a data repository called bugs.

The way this tools works is :-

- *the tool automatically identifies fixes by looking into log messages attached in the bug report.*
- *it then extracts the bug post-fix and bug pre-fix versions of the program. it builds those versions and runs tests on them to collect test data.*
- *bugs that do not meet the pre-requisite of bug localization tool are associated with their metadata which are formed using the class and methods which match the keyword with th bug.*
- *All the versions of the assembled together.*

This method improves on the earlier version where it does verify both the pre-fix and post-fix version of the software before associating it with a file. The major contribution of this paper is to create a open-source training tool for future bug localization tool. It does not attempt to improve the accuracy of the bug localization *in previous data. This work could be used by specific organizations to build and test their own bug localization tools. The iBUGS repository contributed by the paper has also contributed a large number of real software bugs.*

5.3 Bug localization using latent dirichlet allocation

This paper was published by Lukins et al. in 2009. This is one of first papers that has investigated about suitability of information retrieval(IR) technique latent dirichlet allocation(LDA) on open-source software systems of varying size. The authors have presented five case studies to determine accuracy and scalability of LDA based technique and have associated their suitability for use with varying size of software systems. To build a LDA model, the authors take a two step model:

- *first they build a document collection from the source code by extracting semantic information such as comments and identifiers from each source code element at desired level like at class level, method level, etc. which is then used to pre-process the data using techniques of normalization, stemming.*
- *to perform the LDA analysis, the authors use the open source LDA analysis called GibbsLDA++. GibbsLDA++ uses Gibbs sampling technique to estimate topics from the document collection.*

The model expects below parameters to be set before starting the LDA analysis:-

- *the number of topics*
- *the number of iterations for Gibbs sampling process.*
- *amount of smoothening desired for topic distributions per document, represented by α.*
- *the smoothening factor for word distributions per document represented by β.*

The LDA analysis results in :

- *the word-topic probability distribution, and*
- *the topic document distribution.*

The accuracy of approach is defined in terms of rank of query results on a single bug query. The experiment is run for many iterations and the average rank is displayed as the actual output.

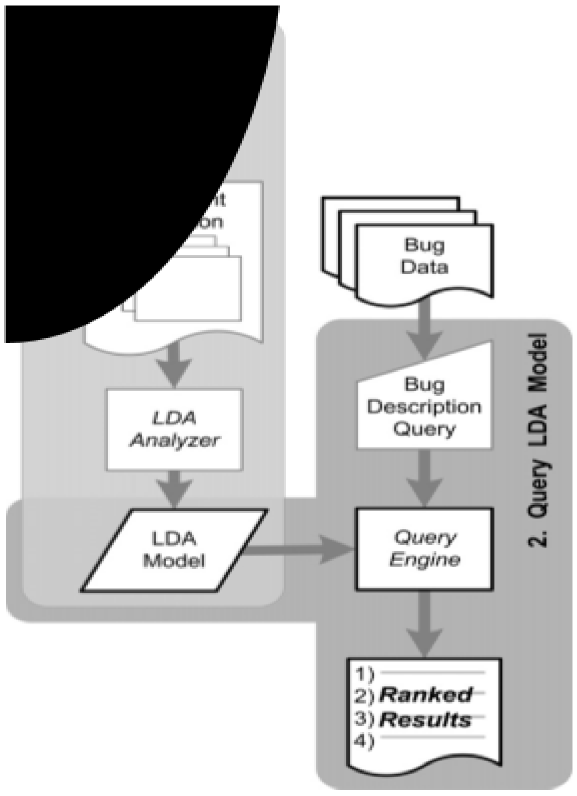


Figure 1: *The below diagram has been taken from the paper under discussion. It explains the LDA process discussed above.*

Case 1 examines whether the accuracy of LDA based bug localization is better than LSI by conducting experiment on benchmarked data from previous LSI experiments. Case 2 examines the accuracy of LDA based technique over exhaustive list of bugs raised over a period of a software release. Case 3 expands the determination of the accuracy of the LDA based technique to bugs from two different software systems. Case 4 and Case 5examines the LDA based approach vs software size and source code complexity respectively.

Unlike the papers discussed above, this paper has done a exhaustive survey of LDA based approach on bug localization. This paper has improved the work done in previous papers by taking into consideration the software size, bugs from entire software development lifecycle raised over a release which would include bugs from unit, system, integration and regression tests.

The paper infers that there was no relationship between Software Development Index metrics and LDA based bug localization techniques. The paper also infers that size of software does not affect the LDA based bug localization technique which means that LDA based technique is suitable for large scale software systems.

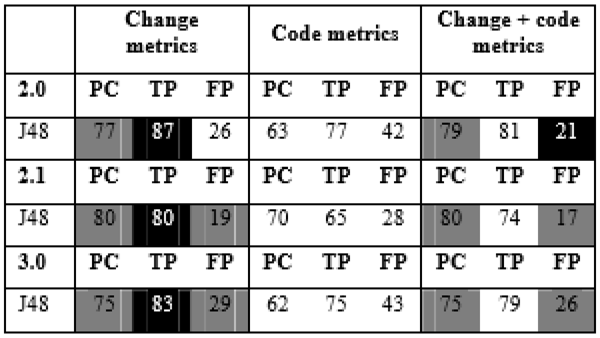
5.4 A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction

This paper [19] was published by Moser et. al in 2008. This paper presents comparison and analysis of the bug localization using two different software metrics: change metrics and code metrics. Classification model were built using Naive Bayes and decision tree method. The work done by this paper is significant as none of the papers earlier have examined use of process related metrics for bug localization.

The authors have used dataset from PROMISE repository. They have applied cost-sensitive classification technique to account for errors made by use of different classification metrics.

The results for cost factor of 0.5 show that defect predictors based on change data outperform significantly those based on static code attributes. The left most column is different versions of J48 model. Correctly classified instances(PC), True Positives(TP) and False Positives(FP) are referred to in figure when J48 model was run on decision tree.

Figure 2*:- The above figure was taken from the paper under discussion.*



	Change metrics			Code metrics			Change + code metrics		
2.0	**PC**	**TP**	**FP**	**PC**	**TP**	**FP**	**PC**	**TP**	**FP**
J48	77	87	26	63	77	42	79	81	21
2.1	**PC**	**TP**	**FP**	**PC**	**TP**	**FP**	**PC**	**TP**	**FP**
J48	80	80	19	70	65	28	80	74	17
3.0	**PC**	**TP**	**FP**	**PC**	**TP**	**FP**	**PC**	**TP**	**FP**
J48	75	83	29	62	75	43	75	79	26

The paper has limitations in form of predicting the results by taking only limited numbers of context variables for tests. So authors are not really confident on generalizing the published results without going for further tests. This paper has added value to study of bug localization by using the process based metrics into the study.

5.5 A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report

This paper was published by Nyugen et al in 2011. The authors have proposed a new bug localization tool called BugScount which is an automated tool to narrow down the search space of a buggy file. This tool is significant because this tool has the ability to produce results with accuracy up to 45% with recommended list of top 10 files.

BugScout uses two artifacts - source code and bug reports. The model has two components for these artifcacts - S-component for Source Code and B component for Bugs. Using LDA, source code is converted in the used S-Component. Text from comments and identifiers in source file are extracted to form the words in the source document s. Each source document has N_s words. Each position is considered to represent a topic vector. Using LDA's machine learning model per topic word distribution is found. This is a generative process based on the change in the files.

The B-Component of the BugScout model is also formed using bug report as a document and using technical keywords in bug report to get the per word distribution from the bug.

To predict the performance of BugScout the authors have used top rank evaluation approach. For the result found, if the correct result is ranked amongst top 20 files, then the model is thought to be performing.

Though this model introduces the concept of rank based results, the model does not take into account historic fixes in the software code which was taken into account in the earlier models. The authors have also limited themselves to very limited systems and programming language. They have tried all their experiments on java based systems.

5.6 Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling

This paper was published by Nyugen et al in 2012. The authors have used IR techniques for finding duplicate bug reports from the database of all the reported bugs. This issue is prevalent in software industry and causes a lot of loss of time investigating two issues with same root cause. The paper introduces DBTM, a duplicate bug detection model which takes advantage of topic based feature selection and IR based feature selection.

As was seen in last few models, DTDM model makes use of vocabulary in bug reports to find create a document and it is used to give the probability of duplicity of the bugs.

This paper is very different from all the papers discussed above since it talks about automating the detection of duplicate bugs using IR approach. So though it does not take our discussion of bug localization any further, it does help in suggests ways to improve bug turn-around time which would be important in any software development procedure.

5.7 Improving Bug Localization using Structured Information Retrieval

This paper was published by Saha et al in 2013. The authors introduce a new tool names BLUiR which has been built on proven open-source IR toolkit named Indri. This tool improves earlier used bug *localization technique for reading source code. In all earlier techniques the source code was read a s a flat text file, ignoring the structure of the code. Though reading the code as a flat text file makes modeling easier, it sacrifices opportunity to exploit this structural information to improve localization accuracy.*

The tool specializes in recognizing and addressing domain specific particulars of bug localization. This method has specifically improved the accuracy of the bug localization over previous discussed automated tool BugScout. The authors have also compared the tools with other proposed tools like BugLocator, Zhou and have reported that BLUiR has performed better than the rest of them consistently.

5.8 Enabling improved IR-based feature location

This paper was published in 2014 by Binkley et al.[22]. This paper takes forward the idea of use of IR for bug localization to feature allocation. This paper used VSM model as retrieval model and represents search vector as a vector of term over the entire vocabulary. This model would add a certain weight to the terms that have been frequently used in the documents thus helping them give more accurate results compared to any other models discussed in paper.

However, this model takes the code as text so it sacrifices use of code structure to retrieve useful information from the code repository. This paper makes use of SEMARU data repository conducting its tests.

6. DATASETS

The datasets used in the papers are :-

- *Extraction of Bug Localization Benchmarks from History paper contributes iBug repository for open source use*
- *SEMARU dataset is has been used by Enabling improved IR-based feature paper.*

7. AREA OF IMPROVEMENT

Bug localization is critical for reducing bug turn-around time. The work reviewed in this paper has been tested on very small sample of data, the source code used for the code is either written in one coding language or from one company's repository. In today's software industry where companies use a lot of third party software for selling their product, testing bug localization tools on just one dimensional sample of data is not a good proof of their accuracy. Work needs to be done to test all the proposed tools on wide variety of bugs from different companies, different types of organizations like health, web based, etc.

8. CONCLUSION

This paper reviews few techniques for bug localization proposed between 2008 to 2014. Information retrieval based techniques have proved to show promise for bug localization. Efforts are being made to improve the technique with new tools using structure of the code to find the relation between raised defect and software code. There is a scope to extend this work to field of feature localization too. The testing of bug localization tools have be done on a wider range of data to prove their credibility so *that they can be adopted to make some real difference in software development process.*

9. ACKNOWLEDGEMENT

I would like to thank Dr. Tim Menzies, Professor, Computer Science Department, North Carolina State University to give me this opportunity to read so many research papers and understand usefulness of models in software engineering.

10. REFERENCES

[1] F. P. Brooks, Jr. No silver bullet essence and accidents of software engineering. Computer, 20(4):10–19, Apr. 1987.
[2] V.R. Basili and B.T. Perricone. Software Errors and Complexity: An Empirical Investigation. Communications of the ACM, Vol 27, No 1, Jan 1984, pp. 42-52.
[3] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, Tien N. Nguyen. A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report 978-1-4577-1639-3/11/$26.00 c2011 IEEE
[4] Looking For Bugs in All the Right Places - Robert M. Bell, Thomas J. Ostrand, Elaine J. Weyuker
[5] Extraction of Bug Localization Benchmarks from History - Valentin Dallmeier & Thomas Zimmermann
[6] G. Denaro and M. Pezze. An Empirical Evaluation of Fault-Proneness Models. Proc. International Conf on Software Engineering (ICSE2002), Miami, USA, May 2002
[7] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy. Predicting Fault Incidence Using Software Change History. IEEE Trans. on Software Engineering, Vol 26, No. 7, July 2000, pp. 653-661.
[8] T.M. Khoshgoftaar, E.B. Allen, K.S. Kalaichelvan, N. Goel. Early Quality Prediction: A Case Study in Telecommunications. IEEE Software, Jan 1996, pp. 65-71.
[9] T.M. Khoshgoftaar, E.B. Allen, J. Deng. Using Regression Trees to Classify Fault-Prone Software Modules. IEEE Trans. on Reliability, Vol 51, No. 4, Dec 2002, pp. 455-462.
[10] A. Mockus and D.M. Weiss. Predicting Risk of Software Changes. Bell Labs Technical Journal, April-June 2000, pp. 169-180.
[11] A. Marcus, A. Sergeyev, V. Rajlich, J.I. Maletic, An information retrieval approach to concept location in source code, in: Proc. 11th Working Conference on Reverse Engineering, Delft, The Netherlands, November 2004, pp. 214–223.
[12] D. Poshyvanyk, Y.G. Guéhéneuc, A. Marcus, G. Antoniol, V. Rajlich, Combining probabilistic ranking and latent semantic indexing for feature location, in: Proc. 14th IEEE Int. Conf. on Program Comprehension, Athens, Greece, June 2006, pp. 137–148.
[13] D. Poshyvanyk, Y.G. Guéhéneuc, A. Marcus, G. Antoniol, V. Rajlich, Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval, IEEE Trans. Softw. Eng. 33 (6) (2007) 420–432.
[14] T. Hofmann, Probabilistic latent semantic indexing, in: Proc. 22nd Annu. ACM SIGIR Int. Conf. on Research and Development in Information Retrieval, Berkeley, CA, USA, August 1999, pp. 50–57.
[15] [20] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval, pages 3–10, 2007.

[16] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. Information and Software Technology,52(9):972 – 990, 2010.
[17] 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 Proceedings of the 2012 International Conference on Software Engineering, ICSE 2012, pages 14–24, Piscataway, NJ, USA, 2012. IEEE Press.
[18] P. McCullagh and J.A. Nelder. Generalized Linear Models, Second Edition, Chapman and Hall, London, 1989.
[19] A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction - Raimund Moser Witold Pedrycz and Giancarlo Succi
[20] Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling - Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, Chen
[21] Improving Bug Localization using Structured Information Retrieval - Ripon K. Saha Matthew Lease Sarfraz Khurshid Dewayne E. Perry
[22] Enabling improved IR-based feature location - Dave Binkley A,Dawn Lawrie A,Christopher Uehlinger A,Daniel Heinz B