ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2015

Automation Framework for Bug Localization using Information Retrieval Techniques

Dhanashree P. Pathak, Srinu Dharavath

Assistant Professor, Dept. of Computer Engineering, GSMCOE, Savitribai Phule Pune University, Maharashtra, India

Assistant Professor, Dept. of Computer Engineering, GSMCOE, Savitribai Phule Pune University, Maharashtra, India

ABSTRACT: Bug Localization is the technique to find the relevant source code entities from a bug report in order to fix that issue. Till date most of bug localization work in industry, at least in the small scale software companies, is done manually. It is a very time consuming activity as it adds on manual efforts to locate the relevant source code. It leads to taking long time to fix the bug which evidently leads to more cost for maintenance. To somewhat lessen this cost one way is to automate the bug localization process in order to somewhat control the time and effort parameters. This strategy has its own limitations but still can show some positivity if we apply information retrieval techniques and consider bug localization as classification problem. This approach will open doors for more research in this area. This paper has suggested some IR techniques as Relational Topic models, Link Importance in Relational Topic Model to achieve the bug localization.

KEYWORDS: Bug Localization; Relational Topic Models; software maintenance cost; software engineering; Link importance; Topic Modeling.

I. INTRODUCTION

Bug localization is a process of identifying the relevant source code in reference with the logged bug report. The general procedure followed in small scale computer software companies is when a tester logs a defect/bug the bug tracking system alerts the development team. It goes through the defect life cycle and at some point the developer takes the bug for resolving it. At this point he goes through that bug , understands, analyse it and figure out which source code files needs to be modified/added/removed in order to fix that bug and acts accordingly. This whole process is manual process. This process can be faster and more efficient if the developer who is working on that bug has prior and proper knowledge of the source code but in many software companies in the maintenance phase the number of developers working on that software product has might been reduced after the development phase. Sometimes the maintenance phases can repeat even after many years of actual development. In software engineering there are proper processes at place such as documentation, proper knowledge transfer so that even if some developer has left the organization then also whoever takes his responsibility can work with same efficiency. But as always there are some differences in theory and practical many times surveys have shown that small scale companies who lack in such documentation and following all software engineering protocols struggle to gain the same efficiency with newly added team members in development team at time of maintenance phase. It takes many man hours to locate the relevant source code entities if bug comes in maintenance phase and the developers who worked on that actual code are not around and no great documentation is available nor the knowledge transfer is done properly.This whole thing again leads to taking much longer time in identifying the relevant source code and fixing it. Even we assume everything is in proper place and all software engineering processes have been followed then also the time a developer needs to identify the relevant source code is more than the automated process.

These difficulties suggest to have some automated framework for bug localization so that overall time for bug fixing is reduced and hence the maintenance cost can be controlled. To make an attempt for automation framework first bug localization problem should be categorized as information retrieval problem as here there is a huge database (Source code files), from this pool of data we are looking for documents (source files) which are relevant with the bug(query). 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. We can use classifier combinations to achieve more efficiency [1].

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 are 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.

Based on our earlier discussion we can put our problem statement as ; Need to develop a automated bug localization system which uses IR techniques as Relational Topic Model, Pre-processing bug reports. This system takes input as a bug report and processes it and as a output lists the relevant source code files.

This paper suggests few of IR (Information Retrieval) techniques which generate a hybrid framework for bug localization. This framework uses the Relational Topic Modeling for modelling source code and then applying proposed algorithm finding the relevant source code entities for a given bug report. Here database maintains a relational topic model and its linked entities. Matrices have been maintained for this. Once the bug is reported it has been pre-processed and later process through the system where the keywords from the bug have been searched in source code and the relevant source code files where you find the maximum matches of the keywords of bug have been considered as list of source code files which are relevant to the bug report. The details of system have been explained in section III.

II. RELATED WORK

Information Retrieval is the study of querying for text within a collection of documents [2]. 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 “Username textbox deactivated”, then an IR model looks for the source code entities which contains the words “username”, “textbox”, etc. (Specifically in this particular system the bug has to be pre-processed and the ‘identifier-text tag/box in UI’ mapping has to be done earlier).

We will see some popular IR techniques and their comparisons for bug localization and related research work by others. Some of the popular techniques are Vector Space Model, Latent Symantec Indexing, Latent Dirichlet Allocation. Here we are focusing on Relational Topic Modeling ,LSI and its variants or LDA and its variants are used.
Latent Symantec 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 [3]. 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.

Latent Dirichlet Allocation - 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.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2015

Along with these IR based techniques pre-processing the bug reports always improves the results. 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 [4] to identify the other entities that also be modified. Removing noises from stack traces and source code pre-processing is also been the area of research for improving the bug localization along with IR techniques.

Due to its theoretical foundation and promising application performance, topic modeling has become a well known text mining method and is widely used in document navigation, clustering, classification and information retrieval.
Given a set of documents, the goal of topic modelling is to discover semantically coherent clusters of correlated words known as topics, which can be further used to represent and summarize the content of documents.

With the wide spread use of online systems, such as academic search engines[5], documents are often hyperlinked together to form a document network. A document network is formally defined as a collection of documents that are connected by links. For example, papers can be connected together via citations, web pages can be linked by hyper-links and tweets can be linked to one another according to the re-tweet relationship.

To take advantage of the link structure of a document network, some link combined topic models such as iTopic[6] have been proposed. However, most existing topic models do not explicitly distinguish the importance of documents on different topics, while in practical situations documents have different degrees of importance on different topics, thus treating them as equally important may inherently hurt the performance of topic modeling. To quantify the importance of documents on different topics, topical ranking (TR) methods [7] can be used, which is extension of basic ranking algorithms such as pagerank[8] and hyperlink-induced topic search(HITS)[9]. Although these ranking methods are initially proposed for the purpose of ranking the web pages, it can also be used to rank other kind of documents, such as research publications cited by each other, since concept and entities in those domains are similar[10]. In this paper, it is focused to build a framework to incorporate link based importance into topic modeling. Specifically topical ranking methods are employed to compute the importance scores of documents over topics, which are then leveraged to guide the topic modeling process.

This paper baselines its research based on RTM using variant of LDA and Link importance in RTM[11].

III. Implementation Details Of System

A. *General Block diagram of Bug Localization System using RTM :*

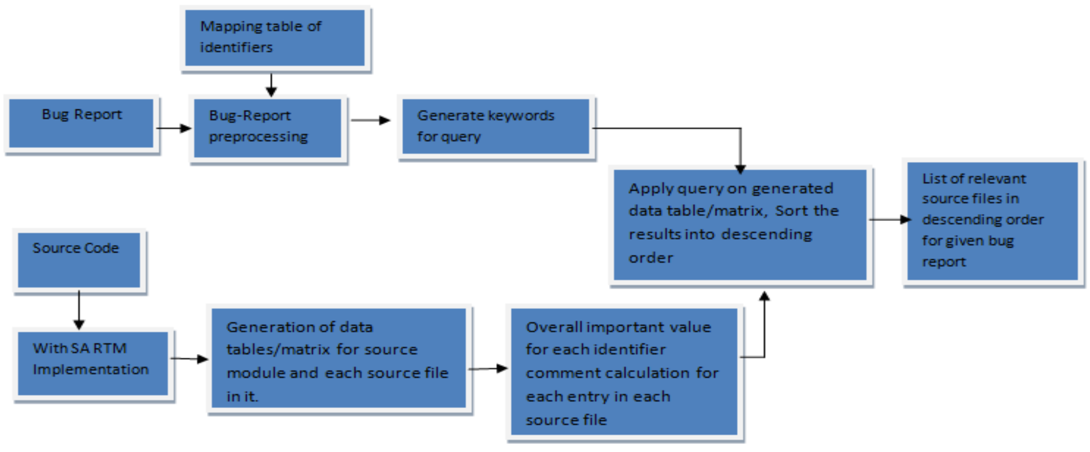
Mapping table of identifiers
Bug Report
Bug-Report preprocessing
Generate keywords for query
Apply query on generated data table/matrix, Sort the results into descending order
List of relevant source files in descending order for given bug report
Source Code
With SA RTM Implementation
Generation of data tables/matrix for source module and each source file in it.
Overall important value for each identifier comment calculation for each entry in each source file

Fig 1. Generalized block diagram of the system

B. *Description of the System:*

Above system uses Relational Topic Model (RTM) with variant of LDA to apply on source code. Topic modeling means, from a given huge set of documents, analyze those documents and try to find out they are discussing what "topic". There are many standard relational topic models are present including LDA. Mainly they give us information about each document is talking about which topic. However, the detail descriptions of standard RTM and LDA is out of scope for this paper.

If company is intending to implement the automated bug localization for further maintenance releases and whole development cycle then at the time of the system design and architecture, the ground work for automation framework for bug localization starts.

In this case, the system architects keep a track of system modules and their inter dependencies and dependent variables and inter relations in separate matrix format. Later this information is used to generate the database of matrices for bug localization. E.g. If a software related to banking is supposed to be developed then there might be different software modules such as customer information, types of accounts, calculation of interest etc. To generate the source code modules the system architects may bundle the calculation of interest under the ***package/module/method*** calculate_interest depending upon the size of the software. Here we know that this module will contain all source code files related to calculation of interest. So the system architects will build a database matrix for each such module in product which will consist of following details;

Source_code_module_name	Source_Code_File1_in_that_module	Comments from that source code file	Identifiers used in the file	References to all other files from this file
Cutomer Module	Companymaster1	** change template, choose Tools | Templates* open template editor.*/ /**@author Administrator*/ /***@return company_id*/	company_id, company_name, company_address, state, cell_no, vat_tin_no,cst pan_card, tan_no	Salesorder, Companymaster

Table I : Source module, source file and identifier, comment matrix

We will find such entry for each module and each source code file which is going to be developed in the product. This lays the ground work for bug localization automation.

Now, in another case, where the source code is already developed and we are trying to automate the bug localization in maintenance phase, for such instances we are using Relational Topic Modeling. Simply we are inversing what we did in above case. Here first, the source code will be analyzed and a Relational Topic Model will be applied to generate what ***"Topics"*** are generated in this developed software. That would be our software ***package/module/method*** as explained above. While generating "Topics" from Relational Topic Modeling , we are using approach similar to LDA. There we are dealing with documents and topics, in same way, here we are reversing the topics and their relevant documents. Exact similar matrix can be used only the difference is here we are targeting documents based on their topics.. These documents are nothing but our ***individual source code files*** and ***"topics"*** from RTM are the ***package/module/method*** . Here we are just taking the source code file names. Now we have source code modules and all source code files so that we can generate the above mentioned matrix.

Once we got this information we can generate a matrix which will keep the database for the count or say importance of a identifier in that source file. If some identifier is the key or it has been referenced in comments as well as in source code file more than once then obviously that must be the key factor for that source code file. So we keep a track of count of every identifier, function that have been referenced/mentioned in comments section how many times in each source file. The generated matrix for this is somewhat like as follows;

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2015

Source_file	Identifier	Count in file	No of references in comment	Comment
Companymaster 1	company_id, Company_name,	5 4	2 1	// fetching company_id from table

Table II : Extension to Table I(Extended columns...partial part of whole table is shown .Actual table contains more details identifiers)

With help of above matrix, ***Overall Importance Weight*** is calculated by considering the no. of occurrences in source file. Later this is useful for finding out the most relevant entities.

The another major part of this system is ‘Bug Pre-processing’. When a tester logs a bug, this system preprocesses it. The detailed algorithm is mentioned in next section. What this algorithm does is, takes the bug report as input and removes all stop words. Only keeps identifiers, comments and essential description. Later it checks for the mapping of identifier to text label matrix in database and gets the identifier name/variable name for the used text label in description/title part. Hence it narrows down the bug to list of keywords which we need to find in source code files. And later the system based on their maximum occurrences in source files leads us to most relevant source code files.

Now, what is the need of this? As we know, in black box testing, mostly the tester is unaware of the details of the source code. He interacts with the system through GUI so he is only aware with the terms provided on the user screens provided by that application. For example in some banking application or General store application, if there is a screen which asks for customer feedback, then it is not going to show the variable name used to store customer feedback on user screen. On user screen the text label may appear as “ Customer Feedback : “ or “Please enter your feedback :” In actual source code file which stores the entered customer feedback, the variable/identifier used might be “Cust_Feedback”. So in source code file value will be assigned to ‘Cust_Feedback’. E.g. the tester has logged a bug saying “ Customer feedback field is inactive. Cannot enter feedback”. To work this system best the tester should follow certain rules as to fill out the mandatory fields of bug report which are usually title, description, severity, relevant module/module area/domain(different names as per different bug tracking systems). Assume that he has filled the necessary details. Title as mentioned above, description as same as title, severity as high and relevant module as “Feedback”. The main task of Bug preprocessing module is from this bug report deduce it till the keyword(identifier or similar thing) which is used in the source files. In this case that would be ‘Cust_Feedback’. So how this is achieved? For this, first the bug will be preprocessed. Removes all stopwords, divides it in tokens. So now we have probable keywords are as Customer, feedback, inactive. From this we will pick only 2, for that we will consider the other field as relevant module/module area/domain, which says ‘Feedback’. So now our top first keyword would be ‘feedback’ and second keyword would be ‘customer’. Usually the ‘relevant module/module area/domain’ are topics from RTM. Now how will we link to ‘feedback’ to ‘Cust_Feedback’ identifier. For that already we have stored the mapping table in database which keeps a mapping of text_labels used on all UI screens and their respective identifiers/variable names. We need to search the text_label “Customer Feedback’ in that table for exact match or partial match. If we succeed then we have got our keyword as identifier ‘Cust_Feedback’ which now we need to search in source files. So the first module we start searching with is ‘Feedback’ as it’s the topic from RTM and its relevant with this bug. We will search all source files from this module by visiting the Table I database matrix. While searching we keep track of ***Overall Importance Weight*** , so it will give the list of all source files which have the identifier ‘Cust_Feedback’ in ascending order. Definitely there will be the source file which stores the ‘Cust_Feedback’ value on top list as it must have mentioned it in comments as well as identifiers. Limitation is that we might get some extra source files which shows relevance with Cust_Feedback but developer is relieved from going to guess and gets a limited stock of files to look at. We use top-k accuracy[12] system to pick files. Here one way is explained to achieve the relevant files , we might do combinations of techniques as instead of taking ‘Feedback’ as topic , topic may be ‘UI’ and this issue can be handled as UI defect. Mostly this works out in coordination with the system architecture, how the system is designed and how it is going to address its modules.

IV. Algorithms Used

A. *Algorithm for The Bug Pre-processing :*

```
Input: Bug Report B.
Step 1: Input the bug report
Step 2: Parameters selected for classification A. Title B. Description.
    Step 3: Using Query Expansion() routine reduce or expand the entities in title and description e.g. if tv is
        mentioned then take tv and television etc.
    Step 4: Remove the log details and stack traces
    Step 5: If it is a child bug then get the details of parent bug and corresponding fixed source files details. Save.
    Step 6: If bug contains crash report then tokenize till method and class in stack trace. Save the details.
    Step 7 : Parse each token through the indentifer-text_label mapping table and search for exact or partial match.
    Step 8: Save the highest matching entities as keywords for this bug query.
    Output: Preprocessed Bug Report Document Bp containing keywords for query to find in source files.
```

B. *Algorithm for Implementing RTM and overall importance weight on source code :*

```
    Input : Source code, keyword for query
    Step 1: Do Topic modeling using own version of LDA(Assume K)
            (This step will find out the topics list in RTM. It means here it will find the
    modules/methods/packages name used in source code. K is the no. of topics i.e. no. of modules suggested by
    system architects)
    Step 2: Generate matrix of modules/methods/packages (topics) and source code files in them (document).
            (This step will generate the Table I first two columns)
    Step 3 : Generate matrix of source code individual file and related details of related identifiers/comments.
            (This step generates the rest of the columns from Table I like topic distribution words in RTM)
    Step 4 : Do topical link analysis
                    step a: Input : Each source file I with topic distribution $\theta_i$ like identifiers, comments
                    step b: Using topical pageRank produce importance vector, importance score(weight) for
                    each identifier, comment in each source file.
                    (This step basically creates the Table II entries for each source file.)
    Step 5: Go through above generated matrix and search it against the keyword found in Bug pre-processing
  module, check all the weights and sort the source file entries based on their counts of references and get the
  ascending order list of relevant source code files.
    Output : List of relevant source code files
```

V. Results

We have obtained the results by comparing the results of manual bug localization verses the results obtained by our proposed automated framework where we have applied the same source code and bugs as used in manual bug localization.

The results must be verified against two parameters as 1. Accuracy of bug localization after using automation framework and 2. Time consumption to obtain bug localization using proposed automation framework.

Table III and Figure 2 explains about the accuracy parameter. Table III shows the samples of bugs input to the system(column1) , its relevant source code files found by the proposed framework(column 2) and the relevant files found by the manual bug automation(column 3). If column 2 and column 3 are equal or if column 2 is having all files from column 3 then we can say 100% accuracy is achieved as we have verified that our proposed system has given the same results as manual bug localization. Figure 3 depicts this in graph. Here we can clearly see that except 2 bugs (bug 108 and bug 111) out of input 10 bugs we have achieved 100% accuracy. It means overall at least 80% accuracy is achieved considering the given Table III.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 6, June 2015

Table IV and Figure 3 explains about the time consumption by the proposed framework and manual bug localization. Table IV shows Bug ID(column1), its relevant source files found by proposed framework and verified by manual bug localization that the results are accurate(column 2), Time required by manual bug localization(column 3) and for the same input time required by proposed framework (column 4). This table gives us the statistical data about exactly how much time is required by manual bug localization and for the bug localization how much time is required by proposed system. Figure 4 depicts this comparison by graph. The graph clearly shows that the time required by proposed framework is considerably low. As the number of bugs increases still the overall time taken by the system will remain low.

Below the Tables and the associated graphs results have shown.

Bug _Id	**Relevant source code file found by automated framework**	**Relevant source code file found by manual bug localization**
101	CustomerMaster, DAOcompanymaster, Supplier_Master	CustomerMaster Supplier_Master
108	RealPurchaseInvoice ,DummyPurchaseInvoice, RealSales	RealSalesReturn RealSales
115	Dumysalesreturn ,Dummy_sales_Vauchar Real_sales_Vauchar	Dumysalesreturn,
120	VATSearch	VATSearch
102	Supplier_Master ,MasterHome	MasterHome
103	Supplier_Master ,AdminLogin,DAOSupplier	Supplier_Master
107	RealSales, Dummy_Stock_Master , DateButton product_master	Dummy_Stock_Master
111	Dummy_sales_Vauchar,DummySale Invoice, DummyPurchaseInvoice	Dummy_sales_Vauchar, Dumysalesreturn, DummySaleInvoice
110	RealSales ,RealSalesReturn1, RealSalesReturn	RealSales, RealSalesReturn
112	Supplier_Master,Product_Master , CustomerMaster	Supplier_Master

Table III : Accuracy measurement table automated vs manual framework

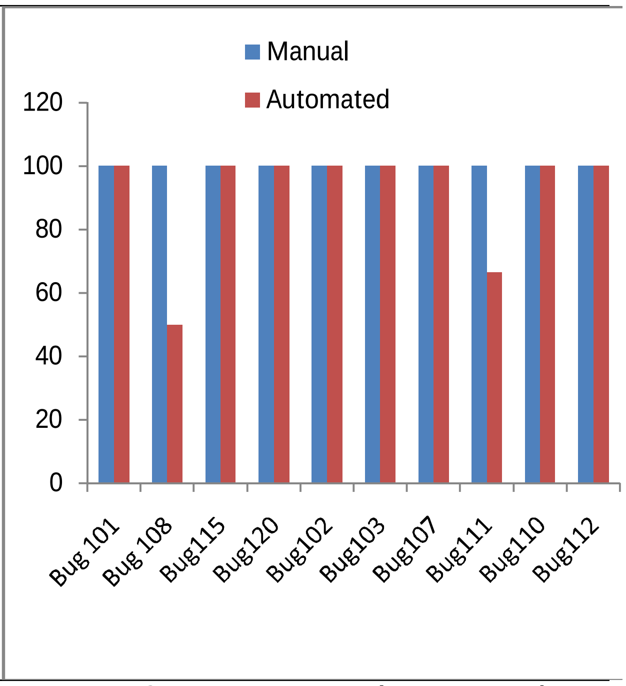

Fig. 3. Time consumption result comparison graph

Bug_Id	Relevant source code file found by manual bug localization and exact same result found by our system	Time Required by Manual bug Localization (Minutes)	Time Required by automated system for bug Localization (Minutes)
101	CustomerMaster.java	3	0.4
110	RealSales.java	3	0.6
120	VATSearch.java	2	0.6
111	Dummy_sales_Vauchar.java	1	0.5

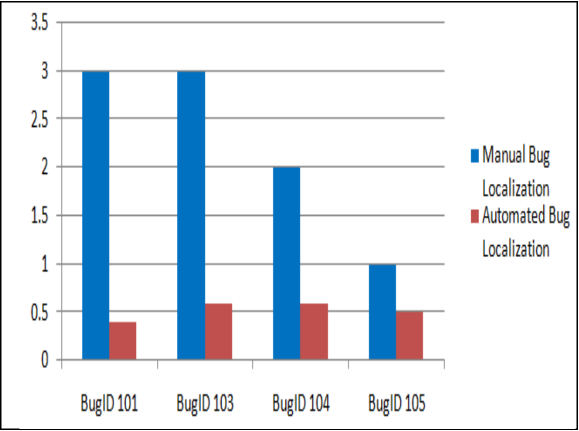

Table III : Time consumption table manual vs automated framework

Fig. 3. Time consumption result comparison graph

*Scale to show tables and data are Bugs: 1:10. *Most of the input bugs are related to UI and basic functionality

Total number of Bugs given as a input to the above automation system : 150
Total number of matching results found from manual and automation framework where both results were same :133
Total % of accuracy of results: 88.66%

VI. CONCLUSION AND FUTURE WORK

The results showed that the proposed algorithm performs better when applied to large no. of bugs in order to get more efficient results based on time and accuracy. Here accuracy might be the same sometimes but definitely we get improvement over time. As the bug fixing time reduces the overall software maintenance cycle time reduced and as the time and efforts reduces, the cost reduces.

As using Relational Topic modeling and its associated proposed algorithms are still under research for bug localization, this system , as of now, comes with limitations, first, it is mostly suitable for small scale software products, whose source code is mid range . e.g. school administration system, online shopping system, store management such types of small scale applications where the development is done through single language, one database is used are perfect candidates for such type of automated frameworks. Like software testing where we could not automate the whole testing process but only 20-30 % can be done and that to mainly for UI and regression, similarly here also we need manual intervention mostly at time of software architecture and design, also needs coordination with bug tracking system. We can use this successfully only on 20-30% of bugs, which are related to mostly UI and functionality to certain extent. Bugs related to logical mishaps are difficult to trace through this system. Considering the versatile nature of big software products it is difficult to apply this framework on that. Today's these limitations lead us for future research to open new doors to conquer these limitations.

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. 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.
4. R. Arnold and S. Bohner, "Impact Analysis—Towards a Framework for Comparison," Proc. Int'l Conf. Software Maintenance, pp. 292-301, 1993.
5. J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and Mining of Academic Social Networks," Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '08), pp. 990-998, 2008.
6. Y. Sun, J. Han, J. Gao, and Y. Yu, "iTopicmodel: Information Network-Integrated Topic Modeling," Proc. Ninth IEEE Int'l Conf.Data Mining (ICDM), pp. 493-502, 2009.
7. L. Nie, B.D. Davison, and X. Qi, "Topical Link Analysis for Web Search," Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR), pp. 91-98, 2006.
8. S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Computer Networks, vol. 30, no. 1-7, pp. 107-117, 1998.
9. J.M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," J. ACM, vol. 46, no. 5, pp. 604-632, 1999
10. N. Ma, J. Guan, and Y. Zhao, "Bringing Pagerank to the Citation Analysis," Information Processing and Management, vol. 44, no. 2,pp. 800-810, 2008.
11. D. Duan, Y. Li, R. Li,R. Zhang, X. Gu, and K. Wen , "LIMTopic: A Framework of Incorporating Link Based Importance into Topic Modeling", IEEE Transactions On Knowledge And Data Engineering, Vol. 26, No. 10, October 2014.
12. D.Pathak, S.Dharavath ,A Survey Paper for Bug Localization, IJSR,Vol.3 Issue 11, 2014

BIOGRAPHY

Dhanashree Prakash Pathak is Asst. Prof at Computer Engineering Dept, GSMCOE, Savitribai Phule Pune University, India. She has 6 years of industrial experience in Software Testing and 6 years of experience in teaching. Received B.E. degree from Walchand College of Engg, Sangli, India 2001.

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