International Journal of Software Engineering and Its Applications
Vol.12, No.2 (2018), pp.17-30
http://dx.doi.org/10.21742/ijseia.2018.12.2.02

A Survey of the Bug Localisation Techniques

Kavya Shree N S[1] and Pushpalatha M N[2]

[1]*Department of Information Science and Engineering*
M S Ramaiah Institute of Technology, Bangalore, India
[2]*Department of Information Science and Engineering*
M S Ramaiah Institute of Technology, Bangalore, India
[1]*kavyashashi.ks@gmail.com*, [2]*pushpalathamn1@gmail.com*

Abstract

Bug localisation is the integral part of software testing and maintenance. Many bug reports are generated during the evolution of software system. The developers must consider the bug reports from various bug tracking systems like Bugzilla, Jira etc. and identify the parts of the source code that contains concerns. It will be a complex task for the developers to identify the source code causing the trouble and it is also time consuming. Many automated bug localisation techniques have been identified in recent years to smoothen the process of bug localisation. There are various static, dynamic, Information Retrieval and hybrid bug localisation techniques. These techniques mainly rely on the similarity between the bug report and the source code. The existing techniques also considers, source code structure, previous bug reports, version histories to improve the efficiency of the bug localisation. This paper aims to provide the short survey on various bug localisation techniques and compares various pros and cons of those techniques. This paper presents and classifies survey of 25 research papers in area of software bug localisation.

1. Introduction

In today's twenty first century bus ticket booking, train ticket booking, online food ordering, groceries shopping, online clothing, attendance, banking all are automated or computerized. Generally, customers trust online systems because it is built on pre-defined functions and hence yields results which are precise and accurate. The main spirit behind any automated system is the software. The software development is not an easy job there are lot of efforts and big mind work of software developers and testers behind it. The effort of building the software is quite agile and it is an extensively increasing process. Software testing plays an ideal role to deliver the correctly working software to the customer. Software testing is an important phase in the software development life cycle where the errors, faults or failures are identified that leads to unintended behavior of the software and make the software bug free. A software bug is flaw in the application software which results in unintended result or output of the program. Software bugs must be identified and removed from the application software to achieve high quality of the software. Testing is highly important because it ensures high software quality, low maintenance costs which are the main parameters for the customers satisfaction. Testing also ensures the software system without failure which otherwise may lead to very complex problems when identified later. Text mining solutions are applied here to perform concern localisation which is part of software testing. Text mining includes concept identification, text extraction, text summarization, text clustering, text characterization etc. The applications of the text mining domain help to solve the key issues related to the concern localisation.

Received (January 15, 2018), Review Result (April 25, 2018), Accepted (May 28, 2018)

ISSN: 1738-9984 IJSEIA

Computer software is a collection or a set of programs that perform specific tasks. From a small digital smart watch to the large complex behemoths which manage space shuttle launches none of the machines will work without programming it. Software program must be checked for error faults or failures before delivering it to the customer to ensure high quality and reliable outcome. Software developers or testers receive bug reports through various issues management tools such as Jira or Bugzilla. If the tester is new to the team it becomes very difficult for him to comprehend the application. Comprehending is the process of thoroughly understanding the application before performing testing. It will be very difficult for him to locate the code blocks that must be modified and hence concern localisation must be automated which in turn decreases the time taken by the tester to identify the code blocks and in this way the maintenance cost is also reduced. This means more bugs can be identified and reduced in shorted life span and thus more bug free software can be delivered to the customer. So, selecting the proper bug localisation tool will save both time and cost and high-quality software can be achieved. By using this, a company can manage resources in a better way and offer solutions much faster. By automating bug localisation companies can utilize resources in better way and achieve solutions much faster.

Considering the above facts incorporating the efficient bug location technique may result in more quality software by considering less time and cost which will be advantageous to the society in their daily lives.

2. Basic Overview of Bug Localisation

Bug localisation is an overall process of location the code unit that contains the bug given in the query. There are four general approaches to perform bug localisation they are: static, dynamic and text retrieval approach and hybrid approach.

Static bug localisation approach makes use of the source code to perform bug localisation. Static method works better because it can be applied on non-working software as well. In most of the static techniques initial method is identified as a start point to search for a feature. Some of the examples of the static bug localisation techniques are Latent Dirichlet Analysis, BLIA *etc*.

Dynamic Bug localisation approached depends on the execution traces of the program. It is difficult to perform this approach if properly working software is not available. This approach makes use of run time data to locate the affected files. The Dynamic bug localisation techniques are often expensive and time consuming when compared to static methods. Some of the examples of dynamic techniques include bug localisation using execution traces, bug localisation using dynamic call graphs *etc*.

Information Retrieval techniques outperforms both static and dynamic based approaches because it can be used in the interactive modes for the refinement of the retrieved results. That is if the developer is not satisfies with the given results (the results may be quite large) about a query then the developer will have the option of altering his query to retrieve more accurate results. During the survey, it was observed that IR techniques where combined with the static and dynamic based approaches to improve the performance of the bug localisation. Some of the examples of the bug localisation technique are BLUiR, Deep Neural Networks an Information Retrieval technique etc. Bug localisation is best performed when combined with static or dynamic approach which is called as hybrid approach.

Some of the key challenges faced during bug localisation using various techniques are listed below.

- Software aging, documentation deficiency, and developer mobility can make software difficult to understand.

- Polysemy refers to words having more than one distinct meaning. Synonmy refers to multiple words have same meaning. Having the problem of polysemy and synonmy will result in less precision during bug localisation.

- The major challenge faced during bug localisation is lexical mismatch which means the words given in the query differs specifically from the words and tokens used in the source code.

3. Survey

Kunrong Chen, Václav Rajlich [1] performed case study on search scenario using dependency graphs. The tool developed could give Abstract System Dependence Graph(ASDG) and a search graph. This method belongs to "Intelligent Assistance" and was evaluated on NCSA Mosaic data set. The result was partial comprehension of the system. Out of 984 functions only 22 were visited.

Chao Liu, Xifeng Yan, Long Fei, Jiawei Han, Samuel P [2] proposed a statistical based technique called SOBER, which doesn't require any prior knowledge of the program. SOBER figures out predicates during both correct and incorrect runs and assigns predicate as bug relevant if the predicate of incorrect run varies from the predicate of the correct run. Author evaluated this approach on siemens suite and could retrieve 68 bugs out of 130 bugs.

Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, Václav Rajlich [3] suggested semi-automatic bug localisation technique SITIR which combines two information sources one is the execution traces of the source code and second is the comments and identifiers included in the source code. Author applied Latent Semantic indexing, an information retrieval method to perform topic modelling. Author integrated dynamic and Information retrieval technique on the JEdit and Eclipse source code. The case study results indicated that SITIR outperformed Latent Semantic Indexing and Scenario Based Probabilistic Ranking. SITIR when applied to Eclipse showed results close to PROMISIER.

Stacy K. Lukins, Nicholas A. Kraft, Letha H. Etzkorn [4] proposed Latent Dirichlet Allocation, a static bug localisation technique. According to the author modularity and extensibility are the main strengths of LDA when compared to LSI and pLSI. Initially an LDA model is generated for the given software structured. Then this model is queried as many times as required until the bug is localised in that software version. The drawback of LDA was this method was intractable for direct computation. This approach was performed on rhino, eclipse and Mozilla data sets. The results clearly indicated that LDA outperformed LSI and vLSI. Out of eight bugs in eclipse and Mozilla LSI could analyze three bugs (37 %) whereas LDA was able to analyze all the 8 bugs efficiently.

Ren Wu at [5] combines both static and dynamic based bug localisation. In this approach author proposes three steps: In first step execution traces collected are used as the text corpus and the methods inside that execution traces are treated as documents. In the second step, omnipresent methods are removed from the corpus by setting the certain threshold. In the third step feature code are treated as the first-class entities and then the identifiers are retrieved from the rest of the source code to generate the trace-by-identifier matrix. Then LDA is applied to generate a topic model. Author evaluates this approach by presenting case study on JHotDraw tool.

Brent D. Nichols [6] suggests an extension to the Latent Semantic Indexing by integrating it with the data in the previous bug reports. Author suggests three phases: taking out the semantic data from the code base using LSI, adding additional data from the previous bugs and then querying the model. Author evaluates his techniques by performing two case studies. In the first case study, he compares the results with manually observed data. In the second case study, he compares his work with the LSI model without

considering the past data. In both the case studies he uses open source Rhino data set. The results of the case studies showed the effectiveness of this method.

Matthew Beard [7] performs research on how effectively can Information Retrieval technique can be applied to perform code clone localisation. Code clone is the process of identifying and removing the duplicate code by using CCFinderX tool. Then IR technique is used to perform concern localisation. This research showed that the efficiency of concern localisation can be improved by considering the code clone process. It was evaluated on two open source projects Rhino, Eclipse. Code Clone cannot be considered for all the projects if code base does not contain duplicate code.

Shivani Rao, Avinash Kak at [8] compared five Information Retrieval Techniques: Unigram Model (UM), Vector Space Model (VSM), Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA), Cluster Based Document Model (CBDM). The main task of these IR models is to locate the bugs in the code base. IR techniques where evaluated on four open source java projects Mozilla, Eclipse, JEdit and Rhino. The major contribution of this paper was the results indicated that simple IR methods such as UM and VSM outperformed LSA, VSM, CBDM.

Bunyamin Sisman, Avinash C. Kak [9] incorporates version histories with IR to improve Bug Localisation. Here author presents two base models one is defect histories and another in modification histories which are stored in versioning tools. These base models are then incorporated with IR model which significantly increases the performance. Author made use of Bayesian reasoning and Divergence from randomness principles algorithm to perform retrieving. The results indicated rise in the Mean Average Precision by 30%.

Sangeeta Lal and Ashish Sureka at [10] suggested a static character n-gram based bug localisation approach. Author evaluates this approach on the two open source data sets JBoss and Apache. The results indicated that use of n-gram approach where advantageous and robust towards noisy data, comparisons etc. The output measured in terms and SCORE and MAP metrices. The median value for the Score metric on JBoss and Apache was 99.03% and 93.70% respectively. The average precision value was 0.9% and 1.0% for JBoss and Apache bug reports respectively.

Phiradet Bangcharoensap, Akinori Ihara, Yasutaka Kamei, Ken-ichi Matsumoto[11] suggests text mining approach that maps textual similarity between the bug report and the source code. code mining ranks files using product metrics, change history mining ranks files based on change process metrics. The results indicated that buggy files which were retrieved using this approach where in top 20 lists of the bug reports. This approach takes long times to process large source code and to identify buggy files. Author also suggests improving the accuracy and performance in the future work.

Emily Hill, Shivani Rao, Avinash Kak [12] showed that there is a relationship between the query nature and the retrieval performance. The author showed the impact of the various stemmers on the localisation techniques. Short queries showed variations in the result and it is difficult for the author to suggest any single stemmer for them. Whereas long queries contain code snippets and hence does not requires stemming for better retrieval. Author suggests KStem performs well for large queries. The results indicated that MStem performed better for all the three types of queries. The results also indicated the success of stemmers depended on the nature of the query.

Steven Davies & Marc Roper [13] suggests improvement to bug localisation techniques by combining Textual information, Stack traces, Similarity of bug reports and number of previous bugs. Here author suggests using simple vector space model in place of complex IR techniques. Author suggests a technique for combining all the sources of information and a low-cost tool to perform localisation. He applied this method across three open source java projects Ant, JMeter, JodaTime consisting 1143 bugs. This method could retrieve top 10 methods ranking 271 to 322.

Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, Dewayne E. Perry [14] suggests BLUiR which considers structural information such as methods and class names on the code base to increase the efficiency of the bug localisation. Author built BLUiR on top of open source tool Indri [31]. Author evaluates this approach on four open source java projects and compares it with state-of-art technique and the results indicated BLUiR outperformed existing techniques.

Dongsun Kim, Yida Tao, Sunghun Kim, Andreas Zeller [15] suggests two phase model which considers information about the bug report for performing bug localisation. In the first phase, the description of the bug is verified whether it is enough to predict the buggy files. If so then the buggy files are retrieved in the second phase based on the information in the bug report. This method was evaluated on Firefox and Core projects. The results showed that almost 70% of the results points to the correct files. The author compared two phase models with other three models – one phase model, the Usual Suspects and BugScout. The results outperformed the existing methods.

Klaus Changsun Youm, June Ahn, Jeongho Kim [16] suggests Bug Localisation with Integrated Analysis (BLIA) an information Retrieval based bug localisation approach. Here author integrates summary of the bug description, stack traces, structured information in the code base and code base change histories to improve the localisation technique. Author also suggests removing the project key words in the source code which are repeated but useless this in turn helps in reducing the computational costs. Author explored this technique on three open source java projects AspectJ, SWT and ZXing. BLIA was compared with existing techniques such as BugLocator, BLUiR, BR Tracer, AmaLgam and the results clearly indicated that BLIA outperforms in terms of Mean Reciprocal Rank and Mean Average Precision metrics.

Shanto Rahman, Kishan Kumar Ganguly, Kazi Sakib [17] introduced an Information Retrieval Technique which identifies the similarities between the bug reports and the source code files. Author also incorporated rVSM with structural information of the source code and the recently changed files to improvise bug localisation. Author insisted on retaining class and method name as they appear frequently in the bug reports. This method was evaluated on three open source projects SWT, ZXing and Guava and the results indicated that it outperformed BugLocator in terms of MRR by 7% and MAP by 8%.

Sanjana Singh, Sandeep K. Singh [18] created a prototype tool based on the research work. Author suggests Mutation based bug localisation. Here Mutation operators are introduced for Exceptional handling and Multithreading that helps in generating mutations for the code units. This concept is only a research approach and mutation operators for exception handling and multithreading resulted in redundant output.

Yukiya Uneno, Osamu Mizuno, Eun-Hye Choi [19] proposes DrewBL where vector space model is used to find the relevance between the bug report and the source code. He also proposed CombBL which combines BugLocator and Bugspots to improve the efficiency of the bug localisation. Form the results, proposed method achieves time efficiency. The author suggests comparing this with other hybrid bug localisation techniques in his future works.

Tanu Sharma, Kapil Sharma, Tapan Sharma [20] proposed Pachinko Allocation Model to Perform bug localisation. This method was evaluated on Rhino and ModeShape data sets. According to the results the PAM model outperformed LDA model by 15%. Here only lexical information is considered. The author suggests using the combination of both lexical and structural data to improve the process of bug localisation.

An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen [21] proposed DNNLOC model which combines Revised Vector Space Model (rVSM) with Deep Neural Network(DNN) to improve the efficiency of the bug localisation. rVSM identifies the textual similarities between the source code and the bug report and collects the features. Now the DNN is used to relate the words in bug reports with the different code tokens and words in the code base. Author applied the technique on the dataset provided by Ye *et al.*, [26] The results

showed that potentially buggy files where retrieved in almost 50% of the cases. In almost 66% of the cases it could retrieve potentially buggy files in top three files.

Steven Davies, Marc Roper & Murray Wood [22] proposed enhancing bug localisation using similarity between bug reports. This method was applied on four open source projects ArgoUML, JabRef, jEdit, muCommander. When evaluated this method was not so effective when implemented alone but found to increase the performance when aggregated with other bug retrieval techniques.

Sai Zhang Cohngle Zhang [23] suggests solution for bug localisation based on Markov Logic. Morkov Logic is derived from machine learning. Here the model is generated using Markov Logic, with predicates and negations. Each statement in the code base is assigned with the predicate si and if that predicate is true then that statement is treated as buggy. Then the marginal distribution p(si) is calculated to prioritise the buggy statements. Markov logic allows to combine different sources of information like program structure information, prior bug knowledge, statement knowledge and apply in bug localisation. The Markov Approach was implemented to build the tool MLNDebbugger to address bug localisation. This tool was used to evaluate small four programs and results showed that it performed better than the previous approach.

X. Ye, R. Bunescu, and C. Liu [24] suggested learning-to-rank approach that involves developers in emulating bug localisation. This approach considers domain knowledge such as API specification, syntactic structure of the code base and summary about the issues. Author evaluated this approach on six open source java project AspectJ, Birt, Eclipse, JDT, SWT, Tomcat and the results indicated that it outperformed state-of-art approaches, BugScout and BugLocator.

4. Table of Comparison

Below table gives the comparison of various bug localisation techniques which considers techniques used, data set used, Methods, Merits and Demerits. The papers are listed year wise for easy reference.

Table 1. Comparison of various Bug Localisation Techniques

Paper	Year	Technique	Data Set	Methods, Merits	Remarks, Demerits
Kunrong Chen, Vaclav Rajlich [1]	2000	Software Dependence Graph	NCSA Mosaic	Author Performed case study on search scenario using dependency graphs. The tool developed could give Abstract System Dependence Graph (ASDG) and a search graph.	The result was partial comprehension of the system. Out of 984 functions only 22 where visited.
Chao Liu, Xifeng Yan, Long Fei, Jiawei Han, Samuel P [2]	2005	Static based Method – SOBER	Siemens suite	Author suggested SOBER which is statistical based technique. The main advantage of this method is the user does not have prior knowledge on the programs.	This method was evaluated on small projects so its method cannot be generalised on large-scale projects.

Denys Poshyvanyk, Yann-Gae¨l Gue´he´neuc, Andrian Marcus at [25]	2007	PROMESIR Technique was used for feature location.	Eclipse and Mozilla	In this paper author merged both IR technique- Latent Semantic Indexing (LSI) with Scenario based Probabilistic Ranking(SPR) which is a dynamic approach. The results indicated proposed approach is better when compared with individual techniques.	Concern localisation using PROMESIR yields accurate and faster results but the major drawback was the technology behind PROMISIER is computationally intensive.
Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, Václav Rajlich at [3]	2007	Information Retrieval Approach – Latent Semantic Indexing	JEdit and Eclipse	Author suggests SITIR which is semi-automatic bug localisation technique. He suggests combing execution traces along with the comments information in the source code to perform bug localisation.	Author is not sure on selection of scenarios in AspectJ as he is not well versed in that data ser. The results may vary on selection of different scenarios. The results are sensitive towards the query given by the programmer.
Stacy K. Lukins, Nicholas A. Kraft, Letha H. Etzkorn at [4]	2008	Information Retrieval Technique – Latent Dirichlet Allocation (LDA) a Static Approach.	Eclipse	Latent Dirichlet allocation can be used effectively for source code retrieval for bug localisation. The results outperformed LSI approach. When compared using bugs as in [25] LSI could retrieve 3 bugs out of top 8 bugs (36.5 %) where as LDA was able to retrieve all the 8 bugs (100%) out of top 10 bugs. LDA performed 77% when applied to all bugs.	It only used the eclipse bug report for experimentation. This technique must be applied for different data sets and to improve the technique in those cases where LDA dint perform well.
Ren Wu at [5]		Static and Dynamic method combined – LDA	JHotDraw	Author suggests combining both static and dynamic based techniques. He makes uses of execution traces and removing the omnipresent methods from the source code to generate the trace by identifier matrix.	Author suggests extending his work by combing dynamic techniques along with the structural model and conducting systematic study to choose the parameters for the proposed approach.
Brent D. Nichols at [6]	2010	Information Retrieval Model – Latent Semantic Indexing	Rhino	Author suggests an extension to the Latent Semantic Indexing by integrating it with the data in the previous bug reports. Author suggests three phases: taking out the semantic data from	Author suggests extending his work on the large datasets and on different languages.

				the code base using LSI, adding additional data from the previous bugs and then querying the model.	
Matthew Beard at [7]	2011	Information retrieval model LSI and LDA is used with code clones	Rhino, Eclipse	Here author introduced Code Clone. Code clone is the process of identifying and removing the duplicate code by using CCFinderX tool. Then IR technique is used to perform concern localisation. This research showed that the efficiency of concern localisation can be improved by considering the code clone process.	Code Clone cannot be considered for all the projects if code base does not contain duplicate code.
Shivani Rao, Avinash Kak [8]	2011	IR Techniques, Unigram Model (UM), Vector Space Model (VSM), Latent Semantic Analysis Model (LSA), Latent Dirichelet Allocation (LDA), Cluster Based Document Model (CBDM)	Mozilla, Eclipse, Rhino and JEdit	Experimental Results indicated that IR methods are effective as static and dynamic techniques.	According to the results it was seen that simple IR methods like UM and VSM outperformed LSA, LDA and CBDM. Author also suggests integrating IR techniques with Dynamic techniques to achieve high accuracy.
Bunyamin Sisman, Avinash C.Kak [9]	2012	Information Retrieval Model - Bayesian reasoning and Divergence from randomness principles algorithm	AspectJ	Author suggests incorporating version histories and modification history along with the information retrieval model to increase the performance. Bayesian reasoning and Divergence from randomness principles algorithm are used to perform retrieving.	The efficiency of this technique relies on the quality of the bug summary. The evaluation of this approach is performed only on AspectJ dataset and accuracy of results on other projects is open for debate.

Sangeeta Lal and Ashish Sureka at [10]	2012	Static N-Gram based Information Retrieval Model	JBoss and Apache	Author suggested a static character n-gram based bug localisation approach.	
Phiradet Bangcharoensap, Akinori Ihara, Yasutaka Kamei, Ken-ichi Matsumoto at [11]	2012	Text mining, code mining and change history mining	Eclipse	Here text mining approach maps textual similarity between the bug report and the source code, code mining ranks files using product metrics, change history mining ranks files based on change process metrics. The results indicated that buggy files which were retrieved using this approach where in top 20 lists of the bug reports.	This approach takes long times to process large source code and to identify buggy files. Author also suggests improving the accuracy and performance in the future work.
Emily Hill, Shivani Rao, Avinash Kak at [12]	2012	Information Retrieval Technique-Vector Space Model for concern Localisation and Unigram Model for bug localisation	Eclipse	Stemmers where used for performing concern localisation and bug localisation on the java source code. Experiments clearly indicated that there is relationship between the nature of the queries and the retrieval performance.	Overall search improvement was only 1-3% overall. Stemming can have large improvement only for individual queries.
Steven Davies & Marc Roper at [13]	2013	Information retrieval techniques are combined with multiple sources of information	ECLIPSE	Here existing IR techniques are improved by considering many sources of data like classes or methods obtained from similar bug reports, no of bugs associated with methods, and stack traces in the bug reports. The results indicated that by combining this three information the system could perform over 19% compared to existing systems.	Many bug reports will not contain stack traces and hence in some cases it cannot be considered with information retrieval methods.
Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, Dewayne E. Perry [14]	2013	Information retrieval Model – BLUiR Model	SWT, Eclipse, AspectJ, ZXing	Author uses open source tool Indri [31] to build his tool BLUiR. He suggests performance of the bug retrieval can be improved by considering the method names and the class names of the source code.	This method relies on the similarity between the bug report and the code base so if there is any poor coding then it will be difficult to perform bug localisation.

Dongsun Kim, Yida Tao, Sunghun Kim, Andreas Zeller at [15]	2013	Two Phase Model	Firefox and Core	Author presents two phase model. In the first phase contents of the bug report is verified if it is enough to proceed further to the second phase else the first phase is terminated. In the second phase the buggy files are identified based on the contents of the bug reports.	Bug localisation was applicable only on open source projects so it might not be generalizable to close source projects.
Klaus Changsun Youm, June Ahn, Jeongho Kim at [16]	2015	BLIA (Bug Localisation with Integrated Analysis) – a Static approach	ZXing, AspectJ, SWT	BLIA is an information retrieval based approach which considers stack traces of the fault description, source file textual information and change remarks of the code units. The results indicated clear improvements in the metrics when compared with the existing techniques: BLUir:23%, BRTracer:18%, BugLocator-33%, Amalgam- 7%.	This method may not be approachable to industrial projects.
Shanto Rahman, Kishan Kumar Ganguly, Kazi Sakib at [17]	2015	Information Retrieval Model - Modified Revised Vector Space Model (MrVSM)	Guava, SWT, ZXing	Here Modified Vector Space Model is derived which considers similarity between fault descriptions, source code historical data and structure of the source code. To make the system more reliable classes and methods are given higher priority in the bug reports rather than words. From the experimental results, it outperformed other existing methods with 82.61%	This Method must be applied for industrial projects to evaluate its effectiveness.
Sanjana Singh, Sandeep K. Singh at [18]	2015	Source Code Mutation	Eclipse	Here source code mutation technique is proposed for bug localisation. Mutation is the process of making small changes in the existing source code which results changes in the results as well. Here Mutation operators are introduced for Exceptional handling and Multithreading that helps	This concept is only a research approach and mutation operators for exception handling and multithreading resulted in redundant outputs.

				in generating mutations for the code units.	
Yukiya Uneno, Osamu Mizuno, Eun-Hye Choi at [19]	2016	DrewBL based on Vector Space Model	Benchmark and Ye's datasets	Here author proposes DrewBL where vector space model is used to find the relevance between the bug report and the source code. He also proposed CombBL which combines BugLocator and Bugspots to improve the efficiency of the bug localisation. Form the results, proposed method achieves time efficiency.	The author suggests comparing this with other hybrid bug localisation techniques in his future works.
Tanu Sharma, Kapil Sharma, Tapan Sharma at [20]	2016	Information Retrieval Model – Pachinko Allocation Model	Rhino and ModeShape	Author proposed Pachinko Allocation Model to Perform bug localisation. According to the results the PAM model outperformed LDA model by 15%.	Here only lexical information is considered. The author suggests using the combination of both lexical and structural data to improve the process of bug localisation.
An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen at [21]	2017	Deep Neural Networks is used with Information Retrieval model – Vector Space Model	Eclipse	Here an IR model VSM gathers the textual similarity between code units and bug descriptions. DNN is then applied to learn and to relate terms in bug description to the terms in the code tokens and source code terms. The new model in termed as DNNLOC. The results showed DNN and VSM yield more accurate results when combined rather than individual methodologies.	DNN cannot produce highly accurate results due to small number of dimensions.

5. Conclusion

Bug localisation is the complex and time-consuming task for a developer. This paper presents a review and classification on 25 survey papers in the area of software bug localisation. Four categories of bug localisation techniques have been identified they are

static, dynamic, Information retrieval and hybrid. Based on these identified categories 25 survey papers are systematically organized. Further it was found that Bug localisation was best performed when Information retrieval approaches where combined with other approaches. Even though many approaches are already available for bug localisation there is still deficiency for a technique which can be applied effectively in the real-time environment.

References

[1] K. Chen and V. Rajlich, "Case Study of Feature Location Using Dependence Graph", IEEE, **(2000)**.
[2] C. Liu, X. Yan, L. Fei, J. Han and S. P. Midkiff, "SOBER: Statistical Model-based Bug Localization", ESEC-FSE'05, **(2005)**.
[3] D. Liu, A. Marcus, D. Poshyvanyk and V. Rajlich, "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", ASE'07, Atlanta, Georgia, USA, **(2007)**.
[4] S. K. Lukins, N. A. Kraft and L. H. Etzkorn "Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation", 15th Working Conference on Reverse Engineering, **(2008)**.
[5] R. Wu, "Correlating Features and Code by Dynamic and Semantic Analysis", Shanghai Lixin University of Commerce, Shanghai 201620, China.
[6] B. D. Nichols, "Augmented Bug Localization Using Past Bug Information", ACMSE '10, Oxford, MS, USA, **(2010)**.
[7] M. Beard, "Extending Bug Localization Using Information Retrieval and Code Clone Location Techniques", 18th Working Conference on Reverse Engineering, **(2011)**.
[8] S. Rao and A. Kak, "Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models", 2011 Mining Software Repositories, Honolulu, Hawaii, **(2011)**.
[9] B. Sisman and A. C. Kak, "Incorporating Version Histories in Information Retrieval Based Bug Localization", IEEE, **(2012)**.
[10] S. Lal and A. Sureka, "A Static Technique for Fault Localization Using Character N-Gram Based Information Retrieval Model", Proceedings of ISEC '12, Kanpur, UP, India, **(2012)**.
[11] P. Bangcharoensap, A. Ihara, Y. Kamei and K.-i. Matsumoto, "Locating Source Code to be Fixed based on Initial Bug Reports -A Case Study on the Eclipse Project", Fourth International Workshop on Empirical Software Engineering in Practice, **(2012)**.
[12] E. Hill, S. Rao and A. Kak, "On the Use of Stemming for Concern Location and Bug Localization in Java", IEEE 12th International Working Conference on Source Code Analysis and Manipulation, **(2012)**.
[13] S. Davies and M. Roper, "Bug localisation through diverse sources of information", IEEE, **(2013)**.
[14] R. K. Saha, M. Lease, S. Khurshid and D. E. Perry, "Improving Bug Localization using Structured Information Retrieval", IEEE, **(2013)**.
[15] D. Kim, Y. Tao, S. Kim and A. Zeller, "Where Should We Fix This Bug? A Two-Phase Recommendation Model", IEEE Transactions on Software Engineering, vol. 39, no. 11, **(2013)**.
[16] K. Changsun Youm, J. Ahn and J. Kim, "Bug Localization Based on Code Change Histories and Bug Reports", Asia-Pacific Software Engineering Conference, **(2015)**.
[17] S. Rahman, K. Kumar Ganguly and K. Sakib, "An Improved Bug Localization using Structured Information Retrieval and Version History", International conference on Computer and Information Technology (ICCIT), **(2015)**.
[18] S. Singh and S. K. Singh, "A Novel Approach for Bug Localization for Exception Handling and Multithreading Through Mutation", IEEE INDICON, **(2015)**.
[19] Y. Uneno, O. Mizuno and E.-H. Choi, "Using a Distributed Representation of Words in Localizing Relevant Files for Bug Reports", IEEE International Conference on Software Quality, Reliability and Security, **(2016)**.
[20] T. Sharma, K. Sharma and T. Sharma, "Software bug localization using Pachinko Allocation Model", IEEE, **(2016)**.
[21] A. Ngoc Lam, A. Tuan Nguyen and H. Anh Nguyen, "Bug Localization with Combination of Deep Learning and Information Retrieval", IEEE 25th International Conference on Program Comprehension(ICPC), **(2017)**.
[22] S. Davies, M. Roper and M. Wood, "Using bug report similarity to enhance bug localisation", 19th Working Conference on Reverse Engineering, **(2012)**.
[23] S. Zhang Congle Zhang, "Software Bug Localization with Markov Logic", ICSE Companion 14, May, Hyderabad, India, **(2014)**.
[24] X. Ye, R. Bunescu and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge", in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE, ACM, **(2014)**, pp. 689-699.
[25] D. Poshyvanyk, Y.-G. Gue´he´neuc, A. Marcus, G. Antoniol and V. Rajlich, "Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval", IEEE Transactions On Software Engineering, vol. 33, no. 6, **(2007)**.

Authors

Kavya Shree N S, she received B.E. degree in Information Science and Engineering from Atria Institute of Technology. She is Pursuing M. Tech in Software Engineering at M.S Ramaiah institute of Technology, Bangalore. Her research interest is in Software Testing and Maintenance.

Pushpalatha M N is a MTech Degree Holder and is serving as an Assistant professor in ISE Department of MSRIT. She is interested in subjects related to Data Mining and Machine Learning.

