International Research Journal of Computer Science (IRJCS)
Issue 5, Volume 2 (May 2015)
ISSN: 2393-9842
www.irjcs.com

A COMPREHENSIVE SURVEY OF VARIOUS TOPIC MODELS USED IN BUG LOCALIZATION

Ritu Sharma
IEC College of Engineering and Technology
Computer science and engineering department

Prof. Devendra Kumar
IEC College of Engineering and Technology
Computer science and engineering department

Abstract – In software engineering a fault is also called as a bug which leads to a program failure or generates wrong results. The problem can be caused by various factors such as wrong logic, insufficient effort etc. Bug localization technique plays a very vital role in order to find such errors and reduce the effort and time in order to maintain software. The aim of this survey is to look at different existing bug localization techniques so that it can help a software tester to recognize bugs in source code within less time and budget.

Keywords – Fault, Bug localization, Bug localization techniques, Topic models, IR models

I. INTRODUCTION

Software plays an important role in business dealings as well as our daily life. In large software systems; improper-documentation makes software project hard to understand and ultimately leads to a difficult software maintenance task. In order to overcome this, proper testing and debugging jobs are needed. Software testing is a process that involves any activity focused at analysing an attribute or system and to make sure that it meets its requisite results. Also it may be defined as the method of executing a program or system with the intention of finding errors. Whereas, debugging is a systematic process of finding and correcting the number of bugs, or flaws, in a computer program.

A. Information Retrieval

An Information Retrieval system is a software program that stores and manages information on documents, often textual documents but possibly multimedia. The system assists users in finding the information they need. It does not explicitly return information or answer questions. Instead, it informs on the existence and location of documents that might contain the desired information. Some suggested documents will, hopefully, satisfy the user's information need. These documents are called relevant documents [1].

The goal of any IR system is to identify documents relevant to a user's query. In order to do this, an IR system must assume some specific measure of relevance between a document and a query, i.e., an operational definition of a relevant document with respect to a query. A fundamental problem in IR research is thus to formalize the concept of relevance; a different formalization of relevance generally leads to a different retrieval model [5].

B. Topic models

A topic model (or latent topic model or statistical topic model) refers to a model designed to automatically extract topics from a corpus of text documents [2], [15], [17]. A collection of terms that co-occur frequently in the documents of the corpus, for example {mouse, click, drag, right, left} and {user, account, password, authentication} makes a topic. Topic models are algorithms for discovering the main themes that pervade a large and otherwise unstructured collection of documents. Topic models can organize the collection according to the discovered themes.

Topic modeling is a suite of algorithms that aim to discover and annotate large archives of documents with thematic information [2]. Topic modeling algorithms are statistical methods that analyze the words of the original texts to discover the themes that run through them, how those themes are connected to each other, and how they change over time.

Topic modeling algorithms do not require any prior annotations or labeling of the documents, the topics emerge from the analysis of the original texts. Due to the nature of language use, the terms that constitute a topic are often semantically related [2], [7].

The terminology used in topic model is explained below with reference to this figure.

1) Term (word) w_i**:** A string of one or more alphanumeric characters. For example, predicting, bug, there, have, bug and of are all terms. Terms might not be unique in a given document.
2) Document d_i: An ordered set of N terms, $w_1, w_2 \ldots . . w_N$.
3) Corpus :An ordered set of n documents $d_1 \ldots . d_n$.
4) Vocabulary :The unordered set of m unique terms that appear in a corpus.
5) Term-document matrix A**:** An m x n matrix whose ith, jth entry is the weight of term w_i in document d_j.

C. Bug Localization using Information Retrieval

Bug localization is a process of mapping a bug back to the code that might have caused it. Bug and the various stages in its life cycle have been discussed below. Bug tracking system and use of IR models in bug localization has also been discussed below.

Information Retrieval (IR) can be defined as: “Retrieving relevant documents (or documents that satisfy user information need) from large and unstructured collection of documents" [5].
Information Retrieval is an art and science of searching (or retrieving) relevant documents from the large collection of documents.

All these are real world examples that are encountered in daily life. Web search engines such as Google, Yahoo, Bing etc. are the biggest applications of IR system. These search engines indexes millions of documents (unstructured or semi-structured nature) which are used for IR model building. When a user input's a query this IR model is used to provide user ranked list of document which are ordered according to their relevance to the given query [14]. IR models are gaining popularity in bug localization domain mainly because of two reasons: scalability and language independence [4]. These features of IR model allow automated bug localization tools to remain applicable as software grows in size and complexity.

II. TOPIC MODELS IN INFORMATION RETRIEVAL

Topic models were originally developed in the field of natural language processing (NLP) and IR as a means of automatically indexing, searching, clustering and structuring large corpora of unstructured and unlabeled documents. Using topic models, documents can be represented by the topics within them, and thus the entire corpus can be indexed and organized in terms of this discovered semantic structure. Topic models enable a low-dimensional representation of text, which uncovers latent semantic relationships and allows faster analysis on text [3].

A variety of probabilistic topic models have been proposed to analyze the content of documents and the meaning of words [2], [4], [12]. These models all use the same fundamental idea, that a document is a mixture of topics but make slightly different statistical assumptions.

Authors [18] proposed Latent Semantic Indexing (LSI), an indexing and retrieval model that used a mathematical technique called singular value decomposition (SVD) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. Hofmann [18] introduced the probabilistic topic approach to document modeling in his Probabilistic Latent Semantic Indexing method (pLSI; also known as the aspect model).

Latent Dirichlet Allocation (LDA), a popular probabilistic topic model has been proposed by authors [2]. LDA has largely replaced PLSI. One of the reasons it is so popular is because it models each document as a multi-membership mixture of K corpus-wide topics, and each topic as a multi membership mixture of the terms in the corpus vocabulary [10]. This means that there area set of topics that describe the entire corpus, each document can contain more than one of these topics, and each term in the entire repository can be contained in more than one of these topic. Hence, LDA is able to discover a set of ideas or themes that well describe the entire corpus [2].

Authors[2], [4] proposed Hierarchical Topic Model (HLDA)that discovers a tree-like hierarchy of topics within a corpus, where each additional level in the hierarchy is more specific than the previous[6]. For example, a super-topic “user interface” might have sub-topics “toolbar” and “mouse events”.

III. BUG LOCALIZATION

As said earlier, Bug localization is a process of locating a bug to the code that might have caused it. Various stages of a bug life cycle have been discussed below. Bug tracking system and use of IR models in bug localization has also been discussed below.

A. Techniques for performing Bug Localization

Basically two types of techniques are used for performing bug localization, one is static and another is dynamic. Static bug localization techniques work on the source code or a static model of the source code, while dynamic bug localization techniques work on execution traces. In static bug localization, neither operational software nor a test case is required. While dynamic bug localization techniques, requires the working software and also the test case that triggers the bug. The major drawback of dynamic technique is that a program or software developed for locating bugs cannot be made language independent [8]. This work focuses on the task of bug localization (locating the bugs in the source code) using topic models of Information Retrieval (IR).

B. Life cycle of a Bug

In software development process, the bug has a life cycle [19]. The bug should go through the life cycle to be closed. A specific life cycle ensures that the process is standardized. The bug attains different states in the life cycle. The different states of a bug can be summarized as follows:

1) New: When the bug is posted for the first time. This means that the bug is not yet approved.
2) Open: The bug is approved as genuine by the lead of the tester.
3) Assign: The bug is assigned to corresponding developer or developer team.
4) Test: Once the bug is fixed by developer, the bug is given to the testing team .
5) Deferred: The bug, changed to deferred state means the bug is expected to be fixed in next releases.
6) Rejected: If the developer feels that the bug is not genuine, the bug is rejected.
7) Duplicate: If the bug is repeated twice or the two bugs mention the same concept of the bug.
8) Verified: Once the bug is fixed and the status is changed to "TEST", the tester tests the bug. If the bug is not present in the software, the bug is approved as fixed and the status is changed to "VERIFIED".
9) Reopened: If the bug still exists even after the bug is fixed by the developer.
10) Closed or Fixed: Once the bug is fixed, it is tested by the tester. If the bug no longer exists in the software, the status is changed to "CLOSED".

C. Bug Tracking System

A bug tracking system or defect tracking system is a software application that is designed to keep track of reported software bugs in software development efforts. It may be regarded as a type of issue tracking system. Many bug tracking systems, such as those used by most open source software projects, allow users to enter bug reports directly.

A major component of a bug tracking system is a database that records facts about known bugs. Facts may include the time a bug was reported, its severity, the erroneous program behaviour, and details on how to reproduce the bug; as well as the identity of the person who reported it and any programmers who may be working on fixing it. Typical bug tracking systems support the concept of the life cycle for a bug which is tracked through status assigned to the bug. Bugzilla is the very first Bug Tracking System developed for tracking bugs in the year 1998. Bugzilla is written in Perl language.

In 2000, Mantis Bug Tracker was introduced written in PHP. Mantis introduced a much nicer user interface than Bugzilla, and offered more customization, including customization of the bug workflow and state transitions.

JIRA, a commercial product launched in 2003 and built in Java, represented another step forward for issue tracking systems, adding additional customization options, and a powerful plug-in architecture. The JIRA platform tends to work best for large enterprise software projects.

In 2006, two similar projects were introduced: Trac and Redmine. Both are open-source project management and issue tracking systems.

IV. BUG LOCALIZATION USING TOPIC MODELS

In recent times, researchers have developed automated static bug localization location techniques [4], [11], [14] using topic models of Information Retrieval (IR) such as Latent Semantic Indexing (LSI) [18], Latent Dirichlet allocation (LDA) [2] and N-Gram [7].

A. Vector Space Model (VSM) based approach to Bug Localization

It is an algebraic model based on the term-document matrix of a corpus [2]. Rows of matrix represent individual terms and columns represent individual document. VSM represents documents by their column vector in the term-document matrix. The similarity factor of two documents is represented by comparing their two vectors. The two documents will only be assumed similar if they contain at least one shared item [15], [16]. VSM has the following parameters: 1) Term Weighting (TW), 2) Similarity Metric (sim)

B. Latent Semantic Indexing (LSI) based approach to Bug Localization

It is an extension to VSM in which singular value decomposition (SVD) is used to project the term-document matrix D; a term-topic matrix T; diagonal matrix S [12]. In LSI, terms occurring in same documents are grouped together into "concepts" or sometimes called "topics"[12], [14].

Here, documents are still represented as column vector, but the vector now has the weights of topics rather than the weights of single terms. It uses the following parameters: 1) Term weighting (TW) – similar to VSM, 2) No. of topics (K) - controls how many topics are kept during the SVD reduction, 3) Similarity Metric (sim) – similar to VSM [13], [18].

C. Latent Dirichlet Allocation (LDA) based approach to Bug localization

In the approach, first an LDA model of source code is created. Then, the built model is queried to map the bugs existing in that source code[3].

There are 2 steps for constructing the LDA model: build the document collection from source code and perform an LDA analysis on the document collection.

Input to the LDA tool consists of the document collection file generated in step1 of the process. Also, set the following parameters before performing LDA analysis using the tool[4], [9].

1) Number of topics
2) Number of iterations for the Gibb's sampling process.
3) α, a parameter, shows topic distribution/ document
4) β, a parameter, shows word distribution/ topic

D. Character N-gram based approach to Bug Localization

This approach uses the low-level character-level representation. In this approach a relevant document(s) search task for a given query and tests the application of character-level-N-gram based textual features [1]. Thereafter, the proposed model has been implemented and its performance is evaluated. The model comprises of 3 components [7], [11]:

1) Character-N-gram based feature extractor – extracts all the character N-gram from error reports and source code files.
2) Similarity computation – calculates the final score between fault reports and source code.
3) Rank generator – generates the rank list of source code files.

V. CONCLUSION

Bug localization can profoundly reduce the time and effort required to maintain the software. In this paper we have analysed the various approach that can be applied to bug localization. Selection of any bug localization methodology needs a very specific knowledge regarding the programming skills. This survey is an attempt to provide users and testers with the existing bug localization techniques and help them to choose the model on the basis of static and dynamic approach of bug localization. Also, we can say that combining multiple models with different techniques for bug localization is likely to perform better than any single technique alone.

ACKNOWLEDGMENT

We wish to acknowledge Mr. Ankur Sharma (senior software developer, CA, USA) for his constant support and constructive criticism throughout this work. His valuable experience proved to be very beneficial for us in order to complete this work.

REFERENCES

[1] Hiemstra, D. (2009 Retrieval Models. In A. Goker, & J. Davies, Information Retrieval: Searching in the 21st century. John Wiley and Sons, Ltd.
[2] D. M. Blei, A.). Information Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation," Journal of Machine Learning Research, vol. 3, pp. 993-1022, 2003.
[3] G. Maskeri, S. Sarkar and K. Heafield, "Mining Business Topics in Source Code using Latent Dirichlet Allocation", ISEC '08 Proceedings of the 1st India software engineering conference, pp. 113-120.
[4] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation," Proc. 15th Working Conf. Reverse Engineering (WCRE 2008), pp. 155-167, 2008.
[5] M. Beard, N. Kraft, L. Etzkorn and S. Lukins, "Measuring the Accuracy of Information Retrieval," Proc. 15th Working Conf. Reverse Engineering (WCRE 2008), Limerick, pp. 124 – 128, 2008.
[6] S. W. Thomas, "Mining Software Repositories with Topic Models," Proc. 33rd International Conference on Software Engineering (Doctoral Symposium), pp. 1138-1139, 2011.
[7] X. Wang, A. McCallum and X. Wei, "Topical N-grams: Phrase and Topic Discovery," Proc. ICDM'07 Proceedings of the 2007 Seveth IEEE International Conference on Data Mining, pp. 697-702.
[8] J. Zhou, H. Zang and D. Lo, "Where Should the Bugs Be Fixed?," Proc. ICSE 2012 Proceedings of the 2012 International Conference on Software Engineering , pp. 14-24, 2008.
[9] P. Shao, R. k. Smith, N. A. Kraft, T. Atkinson, J. C. Carver and A. S. Parrish, "Combining Information Retrieval Modules And Structural Information For Source Code Bug Localization And Feature Location," Ph.D. dissertation, Department of Computer Science, University of Alabama, 2011.
[10] P. Singh and S. Batra, "A Novel Technique for Call Graph Reduction for Bug Localization," International Journal of Computer Applications (0975 – 888)Volume 47– No.15, June 2012.

[11] 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, February 2012.
[12] C. J. Hayes, B. Nichols,N. A. Kraft and M.D. Anderson,"Improving LSI-Based Bug Localization using Historical Patch data,"The University of Alabama McNair Journal .
[13] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R.Harshman, "Indexing by Latent Semantic Analysis," Journal Of The American Society For Information Science, 1990.
[14] Sangeeta. (2011). A Static Technique for Bug Localization Using Character N-Gram Based Information Retrieval Model. Indraprastha Institute of Information Technology, Delhi, Delhi.
[15] Sisman, B., & Kak, A. C. (2012). Incorporating Version Histories in Information Retrieval Based Bug Localization. MSR (pp. 50-59). Zurich: IEEE.
[16] Anthes, G. (Dec,2010). Topic models vs. unstructured data. Communications of the ACM, (pp. 16–18,).
[17] Chang, J., & Blei, D. M. (2009). Relational topic models for document networks. Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, 9, pp. 81-89.
[18] Hoffman, T. (1999). Probabilistic Latent Semantic Indexing 22nd International Conference on Research and Development in Information Retrieval., (pp. 50–57).
[19] Rakesh. (n.d.). Bug Life Cycle. Retrieved from www.softwaretestinghelp.com: http://www.softwaretestinghelp.com/?attachment_id=98