ISSN: 0974-2115
www.jchps.com Journal of Chemical and Pharmaceutical Sciences

Novel approach for identifying bugs using text classification and information retrieval

S. Sri Gowthem, M. Sriram, J. Sridhar*
Department of Computer Science and Engineering, Bharath University
***Corresponding author: E-Mail: sridhar.cse@bharathuniv.ac.in**

ABSTRACT

Data mining problems in software engineering that are under dynamic investigation. Software errors continue to be frequent and represent the significant reason for system failures. To start with analyze and understand the bug characteristics, and then grouping of similar bugs. A novel approach that applies data mining stratergies to extract information in large software and exploit such extracted information for bug detection. To understand the bug characteristics, this study proposes applying text classification and information retrieval techniques to automatically classify tens of thousands of bug reports.

KEY WORDS: data mining, bugs, software testing, text classification, information retrieval

1. INTRODUCTION

Software testing is established to find errors by executing source code for improving organization and corrective maintenance. Many automated tools test the source code with the intent of finding error consists of false positives, so testers finds difficult to fix and track the bugs which causes the system fail. If a developer code the project in higher programming language version and one finds a part of the source code in lower version in open source code. While integrating the developer gets the compatibility error between the higher and lower versions of programming languages. So here it plays a major role to know the programming version of the open source code. We proposed a novel method for finding version of programming language. And clustering of similar kind of bugs minimizes the time to eradicate the bug easily. Altogether the main purpose of our approach is to minimize developer's time and effort. It's a data mining approach to software testing.

Data mining is a process of semi automatically analyzing large database to find patterns that are valid, novel, useful and understandable. Software errors continue to be frequent and account for the major causes of system failures. In this proposed work it discovers software bugs through data mining approach in different versions of software. By identifying the versions through mining large amount of version history data base the tester can fix the bug in which version the bug lies. Identifying version of the source code helps us to fix bug, suppose a developer made a change what else does to change further, it also helps to find bugs, if a programmer wants to commit changes, but has missed a related change. Software testing can be defined as the intention of producing failure, uncover errors, in large projects. So we used a data mining techniques to eradicate the bugs easily by mining versions, and clustering a similar error which is a time consuming job. Versioning indicates the programming language packages evolved from the origin of the language. Developers locate the bug and then fix it by changing files related to the bug. Traditionally bugs are identified by examining the output from software execution, performing software inspection, or running static analysis tools. Bugs, its location, version where the bug lies stored in a large database by clustering those bugs the severity of the bug, comments describing bug's effect on the software, instructions of replicating a bug.

Motivation: Several kinds of data mining problems in software engineering that are under dynamic investigation based on three key viewpoints: data sources being mined, tasks being supported, and mining techniques being used. The issue of classifying instances of software failures according to their causes arises in two common scenarios:

- When a immeasurable failures are reported by clients of deployed software and
- When a immeasurable failures are prompted by executing a synthetic test suite.

In both cases, it is possible that huge number of failures fall into a relatively small number of groups, each comprising of failures caused by the same software defect. Thus applying data mining approaches for this software testing. The purpose of testing a program is to determine faults that cause the system to fail rather than demonstrating the program correctness.

Problem definition: The existing approach contains mining in source code for diagnosing faults in particular version of the software. The automated tools which concentrates on faults in program but not on correctness. The problem here to mine different version of programming language which is used to code the particular program and also to diagnose bug in that application. Software errors continue to be frequent and account for the major causes of system failures. First analyze and understand the bug characteristics, and then grouping of similar bugs. A novel approach that applies data mining techniques to extract information in large software and exploit such extracted information for bug detection. To recognize the bug characteristics, this critique recommends applying text classification and information retrieval techniques to automatically categorize tens of thousands of bug reports. Semantic fault is the foremost root cause of bugs.

Proposed work: Our approach is based on data mining approach, which has been used to cluster the bugs and identifying the version of the source code. This approach describes the three modules which are used to fix and track the bug. Clustering module uses metric based partition algorithm to compare the bugs for average similarity for clustering similar type of bugs. If there is no minimum similarity it forms the individual cluster centric and finds the similarities with the remaining bug.

Version analyzing module, here the source code is tokenized and all the keywords are parsed for identifying the version of the open source code. These keywords are transacted and matched with the all version packages of a particular programming language which is a large item set. And the merging module combines these two results with the path name. The following figure.1, depicts the methodology of the system to cluster similar kind of software bugs and finding the version of the source code. The following subsection outlines each modules work in detail.

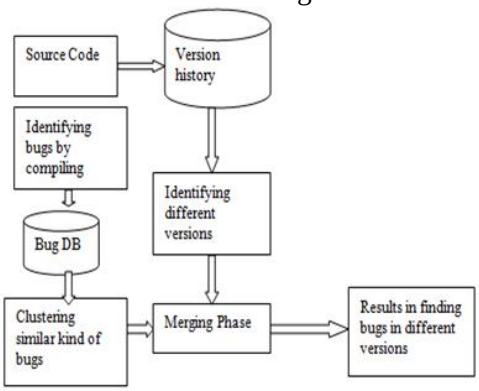

Figure.1. Proposed methodology outline

Procedure used to cluster similar kind of bugs:

Method for Finding Similarity between Bugs (SB): The similarity measured S is bug b_i is a records of bugs stored in database of a set of open source code $s_i \in S$. For each bug $(b_i, b_j) \in S$ is compared with another bug b_j. Represents the similarity between each bug.

$SB(b_i, b_j) = (vb_i^t vb_j) / (|vb_i|_2 * |vb_j|_2)$

Where vd_i and vd_j are the vectors corresponding to the $b_i, b_j \in B(S_i)$ bugs, T denotes the transpose and $|vb_i|_2$ is the length of the vector. For each source code $s \in S$, we have a maximum of $N = S_n^2$ distinct edges between two nodes, where n= |B(s)|. With this conceptual similarity between bugs, we define a set of measure that approximates each cluster elements by measuring the degree to which the bugs in a open source code are related.

Method for Finding Average Similarity of Bugs (ASB) In a Database: The ASB $s \in S$ is $ASB(s) = 1/N * \sum_{i=1}^{N} SB(b_i, b_j)$, where $(b_i, b_j) \in E$, $i \neq j$, $b_i, b_j \in B(s)$, and N is the number of distinct bugs in open source code.

In this view, ASB(s) defines the degree to which bugs of a source code belongs together conceptually and thus it can be used as a basis for computing the clusters.

For a source code similarity of bugs is defined as (b_i, b_j), generates 0 for dissimilar bugs and 1 for similar bugs and forms a cluster in between values for partial similarities between two bugs. Forms the cluster of related bugs like syntactic error, input output error exceptions etc. To better understand the similarity between bugs for clustering, consider a source code $s \in S$ with five bugs after compiling b1, b2, b3, b4, b5. The similarity between the bugs in the source code as shown in the Table 1. From the computation of ASB we consider all pairs of different bugs, thus ASB(s) = 0.5, since the values is positive. Thus the threshold value does not indicate high nor low for forming a cluster of similar bugs but the SB values from Table 1 shows that b1, b3, b2, and b4, b2, b5 and b4 and b5 are closely related respectively that is the ASB between each pair is larger than SB. Thus ASB is not a transitive measure. Clusters of bugs are computerized.

Table.1. Similarities between the bugs in source code s ASB(s) = 0.5.

	b1	**b2**	**b3**	**b4**	**b5**
b1	1	0.32	0.89	0.44	0.48
b2		1	0.18	0.98	0.56
b3			1	0.42	0.25
b4				1	0.89
b5					1

Procedure used to find version: Identifying version of the source code helps us to fix bug, suppose a developer made a change what else does to change further, it also helps to find bugs, if a programmer wants to commit changes, but has missed a related change. Software testing can be defined as the intention of producing failure, uncover errors, in large projects. So we used a data mining techniques to eradicate the bugs easily by mining versions, and clustering a similar error which is a time consuming job. Versioning indicates the programming language packages evolved from the origin of the language. Developers locate the bug and then fix it by changing files related to the bug.

Sequence Pattern Mining Algorithm: Sequence pattern mining is a new-fangled algorithm for finding all frequent sequences within a transactional database. The algorithm is particularly effective when the sequential patterns in the database are long. A depth-first search methodology is used to generate candidate sequences, and different pruning mechanisms are implemented to lessen the search space. In a detailed experimental evaluation using standard benchmark data, we segregate the effects of the individual components of our algorithm. Our performance numbers prove that our algorithm outperforms earlier work by a variable of 3 to over an order of magnitude.

A Pos Tagger (Itemset Phase): POS tagging is typically accomplished by rule-based systems, probabilistic information-driven systems, neural network systems. Because of non-accessibility of statistical information in English, absolutely rule-based frameworks are only able to partly solve the problem of POS tagging. Such frameworks will dispense the vast number of absolutely wrong tagging which would some way or another be available if no constraints were present. The *incomplete* POS tagged for English introduced here decrease the number of possible tagging for a given sentence by daunting some constraints on the sequence of lexical type that can classically occur in an English sentence. On accessibility of statistical information, we can increase our algorithm by including a statistical disambiguation module as a two-layered structure on top of the existing algorithm for performance improvement. The thought, however, is to utilize a rule base for tagging as far as possible, and to take measurable prompts just where there is no other alternative example given in figure.2.

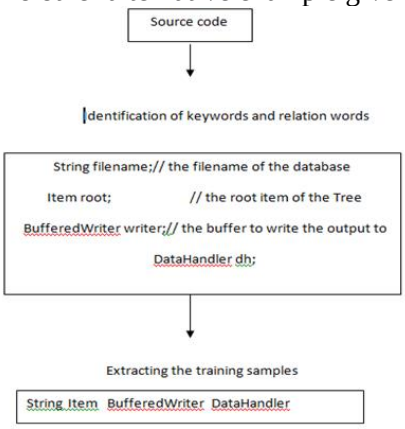

Figure.2.Pre-Processing procedure

2. CONCLUSIONS AND ENHANCEMENTS

This paper presented a novel technology for accumulating a novel result which offers the developer to save time and effort. To extend the compilation, we might want to do further work for following this study: First profoundly analyzing the bugs, we would arrange them according to their semantic traits and explore various technologies. Second we would prone to focus on cost and effectiveness between classifying bugs by analyzing version history. Our mechanism also provides advantages over the other existing strategies whose application requires tweaked and complex runtime environments.

REFERENCES

Achudhan M, Prem Jayakumar M, Mathematical modeling and control of an electrically-heated catalyst, International Journal of Applied Engineering Research, 9 (23), 2014, 23013.

Amir Netz, Surajit Chaudhuri, Jeff Bernhardt, Usama M. Fayyad: Integration of Data Mining with Database Technology. VLDB, 2000, 719-722

Arun k Pujari,Data mining techniques, Tata Mac Grill,2 nd edition Michal Young, Software testing and Analysis, John Wiley and Sons

Benjamin Livshits V, Thomas Zimmermann: DynaMine: finding common error patterns by mining software revision histories. ESEC/SIGSOFT FSE 2005: 296 305.

Calvin Lin, Doc Shankar and Ray Young. Diagnosing Software Bugs and Security Vulnerabilities. IBM Austin. 2004.

Donald Berndt J, John Fisher W, Johnson L, Pinglikar J, Alison Watkins, Breeding Software Test Cases with Genetic Algorithms. HICSS, 2003, 338.

Ducasse S, Rieger M and Demeyer S, A Language Independent Approach for Detecting Duplicated Code, proc. Int'l conf. Software maintenance, pp. 109-118, 1999.

Etzioni O, The World Wide Web: Quagmire or goldmine. Communication of the ACM, 39(11), 1996, 5-68.

Freitas Alex A, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Natural computing series, 2002, 14, 265.

gJason Tsong-Li Wang, Mohammed Javeed Zaki, Hannu Toivonen, Dennis Shasha: Introduction to Data Mining in Bioinformatics. Data Mining in Bioinformatics, 2005, 3-8.

Gopalakrishnan K, Sundeep Aanand J, Udayakumar, R, Electrical properties of doped azopolyester, Middle - East Journal of Scientific Research, 20 (11), 2014, 1402-1412.

Gopinath S, Sundararaj M, Elangovan S, Rathakrishnan E, Mixing characteristics of elliptical and rectangular subsonic jets with swirling co-flow, International Journal of Turbo and Jet Engines, 32 (1), 2015, 73-83.

Ilayaraja K, Ambica A, Spatial distribution of groundwater quality between injambakkam-thiruvanmyiur areas, south east coast of India, Nature Environment and Pollution Technology, 14 (4), 2015, 771-776.

Jagadeesh R.P, Chandra Bose, S. H. Srinivasan: Data Mining Approaches to Software Fault Diagnosis. RIDE 2005: 45-52

Jaiwei Han and Micheline Kamber, Data Mining: Concepts and Techniques (2001), ISBN 1-55860-489-8h

Jinbo Bi, Vladimir Vapnik: Learning with Rigorous Support Vector Machines. COLT 2003: 243-257.

Joel D Martin: Fast and Furious Text Mining, IEEE Data Eng. Bull, 28 (4), 2005, 11-20.

John E. Bentley, Software Testing Fundamentals. Wachovia Bank, Training Solutions Charlotte NC (2004).

Keerthi SS and Shevade SK, SMO Algorithm for Least Squares SVM Formulations, Neural Computation, 15, 2003, 487-507.

Kerana Hanirex D, Kaliyamurthie KP, Kumaravel A, Analysis of improved tdtr algorithm for mining frequent itemsets using dengue virus type 1 dataset: A combined approach, International Journal of Pharma and Bio Sciences, 6 (2), 2015, 288-295.

Li Z, Lu S, Myagmar and Zhou Y, CP-Miner, A Tool for Finding Copy-Paste and Related Bugs in Operating System Code, PROC. SYMP. Operating System Design and Implementation, 2004, 289-302.

Lingeswaran K, Prasad Karamcheti SS, Gopikrishnan M, Ramu G, Preparation and characterization of chemical bath deposited cds thin film for solar cell, Middle - East Journal of Scientific Research, 20 (7), 20141, 812-814, 2014.

Mark Last, Menahem Friedman, Abraham Kandel: The data mining approach to automated software testing. KDD 2003, 388-396.

Mark Last, Menahem Friedman, Abraham Kandel: The data mining approach to automated software testing. KDD 2003: 388-396.

Pan J.Y, Faloutsos C, VideoCube, A new tool for video mining and classification, *ICADL Dec, 2002, Singapore*.

Patrick Francis, David Leon, Melinda Minch, Andy Podgurski: Tree-Based Methods for Classifying Software Failures. ISSRE 2004: 451-462.

Pavel Berkhin: Survey: A Survey on clustering data mining techniques. Accrue software 2(1): (2005)

Premkumar S, Ramu G, Gunasekaran S, Baskar D, Solar industrial process heating associated with thermal energy storage for feed water heating, Middle - East Journal of Scientific Research, 20(11), 2014, 1686-1688.

Raymond Ng T and Jiawei Han member, IEEE computer society. CLARANS: A method for clustering objects for spatial data mining. IEEE transactions on knowledge and data engineering, vol.14, No.5, sept-oct.2002.

Sang Jun Lee, Keng Siau: A review of data mining techniques. Industrial Management and Data Systems 101(1), 2001, 41-46.

Schroeder P.J and Korel B, Black-Box Test Reduction Using Input-Output Analysis. Proc. of ISSTA '00, 173-177, 2000.

Sundar Raj M, Saravanan T, Srinivasan V, Design of silicon-carbide based cascaded multilevel inverter, Middle - East Journal of Scientific Research, 20 (12), 2014, 1785-1791.

Suresh Thummalapenta, Tao Xie: SpotWeb: Detecting Framework Hotspots and Coldspots via Mining Open Source Code on the Web. ASE, 2008, 327-336.

Thooyamani KP, Khanaa V, Udayakumar R, Application of pattern recognition for farsi license plate recognition, Middle - East Journal of Scientific Research, 18 (12), 2013, 1768-1774.

Thooyamani KP, Khanaa V, Udayakumar R, Efficiently measuring denial of service attacks using appropriate metrics, Middle - East Journal of Scientific Research, 20 (12), 2014, 2464-2470.

Thooyamani KP, Khanaa V, Udayakumar R, Partial encryption and partial inference control based disclosure in effective cost cloud, Middle - East Journal of Scientific Research, 20 (12), 2014, 2456-2459.

Thooyamani KP, Khanaa V, Udayakumar R, Using integrated circuits with low power multi bit flip-flops in different approch, Middle - East Journal of Scientific Research, 20 (12), 2014, 2586-2593.

Thooyamani KP, Khanaa V, Udayakumar R, Virtual instrumentation based process of agriculture by automation, Middle - East Journal of Scientific Research, 20 (12), 2014, 2604-2612.

Thooyamani KP, Khanaa V, Udayakumar R, Wide area wireless networks-IETF, Middle - East Journal of Scientific Research, 20 (12), 2014, 2042-2046.

Tristan Denmat, Mireille Ducassé and Olivier Ridoux, Data mining and cross-checking of execution traces: a re-interpretation of Jones, Harrold and Staskotest information, ASE, 2005, 396-399.

Udayakumar R, Kaliyamurthie KP, Khanaa, Thooyamani KP, Data mining a boon: Predictive system for university topper women in academia, World Applied Sciences Journal, 29 (14), 2014, 86-90.

Walawalkar L, Mohammed Yeasin, Anand Narasimhamurthy M, Rajeev Sharma: Support Vector Learning for Gender Classification Using Audio and Visual Cues, IJPRAI, 17 (3), 2003, 417-439.

Zhenmin Li, Yuanyuan Zhou: PR-Miner, automatically extracting implicit programming rules and detecting violations in large software code, ESEC/SIGSOFT FSE, 2005, 306-315.