AN EFFICIENT AND AUTOMATIC BUG TRIAGE FOR MINING SOFTWARE REPOSITTORIES

[1]Salma Shaik, [2]M.Sreeram Murty, [3]J.V Krishna

[1]Research Scholar, [2,3]Associate Professor
Department of Computer Science and Engineering
Sree Vahini Institute of Science and Technology
Tiruvuru, Andhra Pradesh, India

***ABSTRACT*: Bugs are very essential aspects in a software company. The process of fixing bugs is called as a bug triage. Bug triage is an unavoidable step in a software company. In bug triage a correct developer is given to a new bug for fixing it. To manually perform the bug triage is very costly and even time consuming. So text classification techniques are used which uses automatic bug triage. There is a problem of large data i.e. the data should be reduced and the quality of the data should be increased. To perform this instance selection and feature selection are used simultaneously. For this we should know the order for applying instance selection and feature selection, and to know the order we extract the attributes from the bug data sets. For the experiments we are using two open source projects such as eclipse and Mozilla. And our result shows that the data is reduced with high quality bug data sets.**

***Keywords*: Mining Bug repositories, bug data reduction, attribute extraction, instance and feature selection.**

I.INTRODUCTION

In current software expansion, software repositories are large databases for storing the output of software development. Repositories consist of source code, emails, bugs and specification. To manually perform the bug triage is very costly and even time consuming. bug triage. Software projects in a company consist of bug repositories which consist of bug data and it helps developers to handle bug. Updates according to the status of bug fixing. There are two challenges associated to bug data that may influence the effectual use of bug repositories they are huge scale and the low quality of data. Two typical characteristics of low-quality bugs are noise and redundancy. Both of these characteristics affect the bug triage process. So in this paper the two major issues are the large data and low quality. This two issue need to be solved to facilitate the bug handling process. In our work, we combine existing techniques of instance selection and feature selection to simultaneously reduce the bug dimension and the word dimension which improves the quality of the bug data.

A software bug is an fault or failure in a computer program that proves to generate an incorrect or ambiguous result, or to behave in unexpected ways. There are many feasible ways to find bugs in a software. Various Dynamic techniques, such as testing and assertions, depends on the runtime behavior of a program. The most efficient and nice static technique for terminating bugs is a formal evidence of correctness. Bug Patterns are error-free coding trails that arise from the use of erroneous design patterns, misunderstanding of language semantics, or simple and common mistakes. As developers, we many times believe that any bugs in our code must be subtle, unique and require sophisticated tools to uncover. All of the bug pattern detectors are done using BCEL ,which is an open source byte code analysis and instrumentation library. The detectors are executed using the Visitor design pattern; every detector checks every class of the analyzed library or the application. Data mining (the analysis step of the Knowledge Discovery in Databases process, an interdisciplinary subfield of computer science, is the computational process of finding patterns in huge data sets which includes methods at the intersection of machine intelligence and machine learning, statistics, and database systems. The Mining Software Repositories (MSR) analyzes the rich data in software repositories, such as version, mailing list archives, bug tracking systems, control repositories, issue tracking systems etc. to uncover eye catching and function-able information about the software systems, projects and software engineering. By using various data mining techniques, mining software repositories can provide a solution to these problems. A bug repository provides a data based platform to support many types of tasks on bugs, e.g., fault prediction bug localization and reopened bug analysis. In this paper, bug reports in a bug repository are called bug data. bug triage is very time taking method . it includes handling software bugs , which assigns a right developer to a new bug coming In the experiments, we evaluate the data reduction techniques for bug triage on the bug reports of two large open source projects, such as Eclipse. Experimental output shows that by using the instance selection technique to the data set can reduce bug reports but the accuracy of bug triage may be decreased; applying the feature selection technique can reduce words in the bug data and the accuracy can be increased. The bug report consists of matter info concerning the bug and updates associated with standing of bug fixing. Once a bug report is made, a personality's triager assigns this bug to a developer; World Health Organization can try and fix this bug. This developer is recorded in associate item assigned-to. The assigned to will amendment to a different developer if the antecedently assigned developer cannot fix this bug. the method of assigning an accurate developer for fixing the bug is termed bug triage. Bug sorting is one among the foremost time intense step in handling of bugs in software package comes. Manual bug sorting by a personality's triage is time intense and erring since the quantity of daily bugs is massive and lack of information in developers regarding all bugs. Because of all

these things, bug sorting ends up in costly time loss, high price and low accuracy. The information keep in bug reports has 2 main challenges. First of all the massive scale information and second low quality of knowledge as a result of sizable amount of daily reported bugs, the number of bug reports is scaling up within the repository. Noisy and redundant bug's square measures degrading the standard of bug reports. In this paper an efficient bug sorting system is projected which scale will back the bug information to save lots of the labor price of developers. It conjointly aims to create a top quality set of bug data by removing the redundant and non-informative bug reports.

II. RELATED WORK

In this section, we review existing work on modeling bug data, bug triage, and the quality of bug data with defect prediction. To investigate the relationships in bug data, Sandusky et al. form a bug report network to examine the dependency among bug reports. Besides studying relationships among bug reports, Hong et al. build a developer social network to examine the collaboration among developers based on the bug data in Mozilla project. This developer social network is helpful to understand the developer community and the project evolution. By mapping bug priorities to developers, Xuan et al. identify the developer prioritization in open source bug repositories. The developer prioritization can distinguish developers and assist tasks in software maintenance. In our work, we address the problem of data reduction for bug triage. To our knowledge, no existing work has investigated the bug data sets for bug triage. In a related problem, defect prediction, some work has focused on the data quality of software defects. In contrast to multiple-class classification in bug triage, defect prediction is a binary class classification problem, which aims to predict whether a software artifact (e.g., a source code , a class, or a module) contains faults according to the extracted features of the artifact. In software engineering, defect prediction is a kind of work on software metrics. To improve the data quality, Khoshgoftaar et al. and Gao et al. examine the techniques on feature selection to handle imbalanced defect data. Shivaji et al. proposes a framework to examine multiple feature selection algorithms and remove noise features in classification based defect prediction. Besides feature selection in defect prediction, Kim et al. present how to measure the noise resistance in defect prediction and how to detect noise data. Moreover, Bishnu and Bhattacherjee process the defect data with quad tree based k-means clustering to assist defect prediction. In this paper, in contrast to the above work, we address the problem of data reduction for bug triage. Our work can be viewed as an extension of software metrics. In our work, we predict a value for a set of software artifacts while existing work in software metrics predict a value for an individual software artifact.

3. Proposed System The diagram in figure 2 illustrated the system architecture of the proposed system. The input to the system is in the form of bug data set. The bug data set consists all the details of software bugs. Each bug has bug report and the details of the developer who have worked on that respective bug. The bug report is mainly divided in two parts, summary and description. The proposed system gives predicted results in form of output. Basically, there are two types of users in the proposed system. First is the developer and second is the tester. Developer will get software bugs assigned to him. Developer can work on only one software bug at a time. Tester can add new bugs to the system. As shown in figure 2, the proposed system makes use of bug data reduction. In the proposed system, to save the labor cost of developers, the data reduction for bug triage is made. Bug data reduction is applied in phase of data preparation of bug triage. Data reduction mainly has two goals. Firstly, reducing the data scale and secondly, improving the accuracy of bug triage. Techniques of instance selection and feature selection are used for data reduction. Instance selection and feature selection are widely used techniques in data processing. For a given data set in a certain application, instance selection is to obtain a subset of relevant instances (i.e., bug reports in bug data) while feature selection aims to obtain a subset of relevant features (i.e., words in bug data). In the proposed system, the combination of instance selection and feature selection is used.

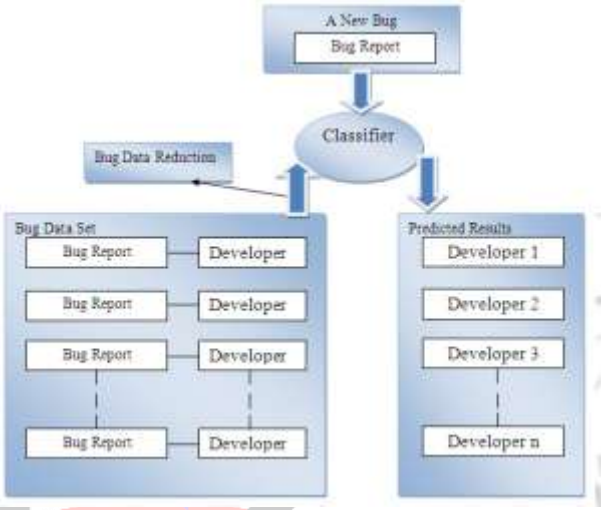

Figure 1: System Architecture

III. SYSTEM ARCHITECTURE

3.1Bug Triage

The aim of bug triage is to assign a developer for bug fixing. Once a developer is assigned to a new bug report he will fix the bug or try to rectify it. In bug report consist of two key items the summary and description about the information of bug which is recorded in natural languages. The summary denotes a general statement for identifying a bug and description denotes the details for reproducing the bug. Fig 1.Shows that before training a classifier with a bug data set. It consists of phase of data reduction.

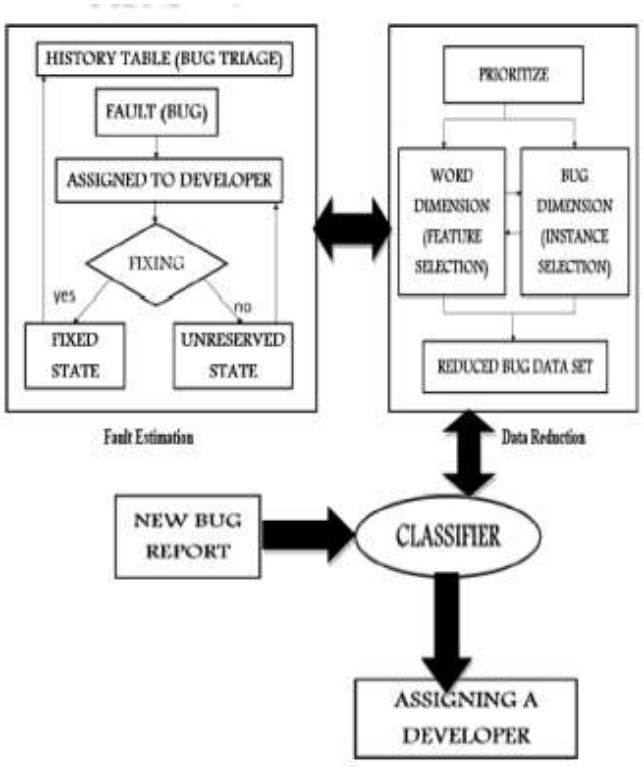

Figure -2: Architecture of Bug Triage

3.2 Data Reduction

The data reduction is used for reduce the scale and improve the quality of data in bug repositories. For data reduction applied as a phase in data preparation of bug triage. By using instance selection and feature selection reduce data, so that get high quality data. For data processing the instance selection and feature selection are widely used. For given data sets instance selection is to obtain a subset of relevant instances that is bug report in bug data and feature selection means to obtain a subset of relevant features that is words in bug data. Instance selection is a technique to reduce the number of instances by removing noisy and redundant instances. An instance selection algorithm can provide a reduced data set by removing non-representative instances .Choose four instance selection algorithms such as Iterative Case Filter (ICF), Learning Vectors Quantization (LVQ), Decrement Reduction Optimization Procedure (DROP) and Patterns by Ordered Projections (POP). Feature selection is a pre-processing technique for selecting a reduced set of features for large-scale data sets. The reduced set is considered as the representative features of the original feature set. Since bug triage is converted into text classification, focus on the feature selection algorithms in text data. Choose four well-performed algorithms in text data and software data such as Information Gain (IG), x2 statistic (CH), Symmetrical Uncertainty attribute evaluation (SU), and Relief-F Attribute selection (RF). Based on feature selection, words in bug reports are sorted according to their feature values and a given number of words with large values are selected as representative features [1][2].

IV. MINING SOFTWARE REPOSITORIES

To understand constantly evolving software systems is a very daunting task. History of software systems are maintained in software repositories. Evolution of software systems are documented by artifacts called Software repositories. They often contain data from years of development of a software project [2]. Examples of software repositories are: Runtime Repositories: Repositories that contain development logs about application usage on deployment sites and useful information of its execution are one of many examples of runtime repositories. Historical Repositories: Bug repositories, source code repositories and archived communication logs are some examples of historical repositories. Code Repositories: Examples of code repositories are Google code and codeforge.net that store source code of various open source projects [3]. A process of software repository analysis which discovers significant and interesting information hidden in these repositories is known as MSR. It processes and analyses the huge software engineering data to detect interesting patterns in this data. It is an open field as in what can be mined and what can be learned from practice. All kinds of software repositories can be mined.

4.1 Software Evolution and Trend Analysis

Analyzing and characterizing how a software system changes over time, or the software evolution [506] of a system, has been of interest to researchers for many years. Both how and why a software system changes can help yield insights into the processes used by a specific software system as well as software development as a whole. To this end, LDA has been applied to several versions of the source code of a system to identify the trends in the topics over time [525, 834, 835]. Trends in source code histories can be measured by changes in the probability of seeing a topic at specific version. When documents pertaining to a particular topic are first added to the system, for example, the topics will experience a spike in overall probability. Researchers have evaluated the effectiveness of such an approach, and found that spikes or drops in a topic's popularity indeed coincided with developer activity mentioned in the release notes and other system documentation, providing evidence that LDA provides a good summary of the software history [832]. LDA has also been applied to the commit log messages in order to see which topics are being worked on by developers at any given time [401, 402]. LDA is applied to all the commit logs in a 30 day period, and then successive periods are linked together using a topic similarity score (i.e., two topics are linked if they share 8 out of their top 10 terms). LDA has also been used to analyze the Common Vulnerabilities and Exposures (CVE) database, which archives vulnerability reports from many different sources [641]. Here, the goal is to find the trends of each vulnerability, in order to see which are increasing and which are decreasing. Research has found that using LDA achieves just as good results as manual analysis on the same dataset. Finally, LDA has recently been used to analyze the topics and trends present in Stack Overflow10, a popular question and answer forum [75]. Doing so allows researchers to quantify how the popularity of certain topics and technologies (e.g.: Git vs. SVN; C++ vs. Java; iPhone vs. Android) is changing over time, bringing new insights for vendors, tool developers, open source projects, practitioners, and other researchers.

4.2 Bug Database Management

As bug databases grow in size, both in terms of the number of bug reports and the number of users and developers, better tools and techniques are needed to help manage their

work flow and content. For example, a semi-automatic bug triaging system would be quite useful for determining which developer should address a given bug report. Researchers have proposed such a technique, based on building an LSI index on the the titles and summaries of the bug reports [7, 41]. After the index is built, various classifiers are used to map each bug report to a developer, trained on previous bug reports and related developers. Research reports that in the best case, this technique can achieve 45% classification accuracy. Other research has tried to determine how easy to read and how focused a bug report is, in an effort to measure the overall quality of a bug database. Here, researchers measured the cohesion of the content of a bug report, by applying LSI to the entire set of bug reports and then calculating a similarity measure on each comment within a single bug report [253, 524]. The researchers compared their metrics to human-generated analysis of the comments and find a high correlation, indicating that their automated method can be used instead of costly human judgements. Many techniques exist to help find duplicate bug reports, and hence reduce the efforts of developers wading through new bug reports. For example, Runeson et al. [737] use VSM to detect duplicates, calling any highly-similar bug reports into question. Developer can then browse the list to determine if any reports are actually duplicates. The authors preprocess the bug reports with many NLP techniques, including synonym expansion and spell correction. Subsequent research also incorporates execution information when calculating the similarity between two bug reports [907]. Other research takes a different approach and trains a discriminative model, using Support Vector Machines, to determine the probability of two bug reports being duplicates of each other [801]. Results are mixed. Finally, recent work has proposed ways to automatically summarize bug reports, based on extracting key technical terms and phrases [540, 709]. Bug summaries are argued to save developers time, although no user studies have been performed.

4.3 Organizing and Searching Software Repositories

To deal with the size and complexity of large-scale software repositories, several IR-based tools have been proposed, in particular tools for organizing and searching such repositories. MUDABlue is an LSI-based tool for organizing large collections of open-source software systems into related groups, called software categories [454]. MUDABlue applies LSI to the identifier names found in each software system and computes the pair wise similarity between whole systems. Studies show that MUDABlue can achieve recall and precision scores above 80%, relative to manually created tags of the systems, which are too costly to scale to the size of typical software repositories. LACT, an LDA-based tool similar to MUDABlue, has recently been shown to be comparable to MUDABlue in precision and recall [844]. Sourcerer is an LDA-based, internet-scale repository crawler for analyzing and searching a large set of software systems. Sourcerer applies LDA and the AuthorTopic model to extract the concepts in source code and the developer contributions in source code, respectively. Sourcerer is shown to be useful for analyzing systems and searching for desired functionality in other systems [528, 529]. S 3 is an LSI-based technique for searching, selecting, and synthesizing existing systems [694]. The technique is intended for developers wishing to find code snippets from an online repository matching their current development needs. The technique builds a dictionary of available API calls and related keywords, based on online documentation. Then, developers can search this dictionary to find related code snippets. LSI is used in conjunction with Apache Lucene to provide the search capability.

V.CONCLUSION

Bug triage is a chip step of computer code maintaining. The projected system aims to form reduced and high-quality bug knowledge in computer code development and maintenance. Processing techniques like instance choice and have choice are used for data reduction. The projected system is used for any open supply comes that generate immense bug knowledge. Various software corporations engaged on comes like banking, food chain management will use the applying of the projected system. The advantage of proposed system is, it combines feature selection with instance selection to decrease the level of bug data sets as well as improve the data quality. The next advantage is, it provide priority according to severity of bug and security so that no another developer can access it.

REFERENCES

[1] Baysal, O., Holmes, R., & Godfrey, M. W. (2012, June), “Revisiting bug triage and resolution practice” In User Evaluation for Software Engineering Researchers (USER), 2012 (pp. 29-30) IEEE.

[2] Jeong, G., Kim, S., & Zimmermann, T. (2009, August), “Improving bug triage with bug tossing graphs” in Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 111-120). ACM.

[3] Park, Jin-woo, Mu-Woong Lee, Jinhan Kim, Seung-won Hwang, and Sunghun Kim, "CosTriage: A Cost-Aware Triage Algorithm for Bug Reporting Systems." In AAAI. 2011.. [4] Kevic, Katja, Sven Christian Muller, Thomas Fritz, and Harald C. Gall. "Collaborative bug triaging using textual similarities and change set analysis", In Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, pp. 17-24. IEEE, 2013..

[5] Xuan, Jifeng, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo "Automatic Bug Triage using SemiSupervised Text Classification" in SEKE, pp. 209-214, 2010..

[6] Alenezi, Mamdouh, Kenneth Magel, and Shadi Banitaan. "Efficient bug triaging using text mining." Journal of Software 8.9 (2013): 2185-2190.

[7] Zou, Weiqin, Yan Hu, Jifeng Xuan, and He Jiang. "Towards training set reduction for bug triage." In Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual, pp. 576-581. IEEE, 2011.

[8] S Hu, Hao, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. "Effective bug triage based on historical bug-fix information." In Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on, pp. 122-132 IEEE, 2014.

[9] Yifan fu · xingquan zhu · bin li‖ a survey on instance selection for active learning‖ 17 march 2012 © springer-verlag london limited 2012

[10] Chengnian sun‖ towards more accurate retrieval of duplicate bug reports‖ http://www.bugzilla.org/
[11] Yoav freund ―experiments with a new boosting algorithm‖ machine learning: proceedings of the thirteenth international conference, 1996
[12] Shay artzi, adam kie _zun ―finding bugs in web applications using dynamic test generation and explicit-state model checking‖ ieee transactions on software engineering, vol. 36, no. 4, july/august 2010
[13] Silvia breu ―information needs in bug reports: improving cooperation .

Authors Profile

SK.Salma M.TECH CSE
Sree Vahini Institute of Science and Technology Tiruvuru Andhra Pradesh.

M. Sreerama Murty
Assoc.Professor SreeVahini Institute of Science
and Technology Tiruvuru A.P.
"M.TECH(CSE)M.TECH(CNIS),(Ph.D)"
Email id :-sreeramsssit@gmail.com

J.V Krishna
Assoc Professor M.TECH of CSE
SreeVahini Institute of Science and Technology Tiruvuru Andhra Pradesh.
Emailid: hodcsesvist234@gmail.com