I J C T A, 9(25), 2016, pp. 69-75 ISSN: 0974-5572

Challenges in Extraction of Data and Analysis of Test cases for Automatic Software Repair

K. Artheeswari*, Saravanan Venkataraman, and M. Dinesh*****

ABSTRACT

In software life cycle bug fixing is a frequent activity. Where large software projects are subject to quality risks of having defective modules that will cause failures during the software execution. Here the software project activity targets at take out the gap between the estimated behavior of a software package and what it truly ensures. This research gap encompasses different malfunctions such as the failure of a program facing to a given scenario. Bug fixing is a task traditionally done by software developers. However, several software repositories contain source code of large projects that are composed of many components. These software repositories include data for the software metrics of these modules and the defective state of each component. In this research work, the challenges faced when automated software repair approach is used to show the attributes that predict the defective state of software components. Software solution architecture is proposed to convert the extracted knowledge into data mining models that can be integrated with the current software project metrics and bugs data in order to enhance the prediction. The results show better prediction capabilities when all the algorithms are combined using subjective returns.

Keywords: Data mining, software testing, defect analysis and data extraction

Software Package development and repair are key modules of software maintenance, which consumes a vast fraction of the total cost of software production [1]. There are many tools available to help with software bug tracing localization authentication and even confirmation generating repairs remains a typically manual, and thus expensive, process. Current software testing trend is clear that there is a persistent need for automatic techniques to supplement manual software development with economical tools. Current research [2] in automated program repair costs by enabling constant program execution in the face of runtime errors using code contracts and specifications to synthesize test case repairs.

The data mining methodology is used to determine many unseen factors regarding software. This includes the success aspects of software projects that attracted researchers for a long time [3], the support of software testing management and the defect pattern discovery. The software defects estimation and prediction processes are used in the analysis of software quality [4]. The quality of software depends on the maturity level of the software development processes [5]. The research work in [6], stressed the importance of assessments to minimize the intensities of defects, especially the requirements and design assessments. They are also used to estimate the required effort and cost of maintenance after delivery. The maintainability evaluation depends mainly on the use of software design metrics. The number of defect densities decreased exponentially in the coding phase because defects were fixed when detected and did not migrate to subsequent phases.

* PhD Research Scholar, Anna University, Chennai, India

** Department of Computer Science, Majmaah University, Al Majmaah, Kingdom of Saudi Arabia

*** College of Computing and Informatics, Saudi Electronic University, Abha, Kingdom of Saudi Arabia

This peer review based assessment process turn out to be very pricy and impractical in the coding phase because the code review process is labor concentrated can be inspected and this effort repeats for all members of the review team [7]. Thus, the prediction of potential defective components will be useful to prioritize which modules are the best candidates for this pricey code reviews, inspection and through testing. This prediction process is a complementary approach that should be used carefully to avoid the trend of leaving blind portions of the system that may contain defects which may be missed. Because assessing software goes toward areas that are believed to be mission critical, several defect gauges based on static code measures are proposed [8]. With the existence of software repositories including [9], several attempts are done to use empirical data to construct and validate different static defect prototypes for multiple software projects or different versions of the same project [10].

Each set includes preparatory text that lists and explains the static methods and other variables of each project. In research work [11] data points communicate to components and their statistic measures as well as a binary variable indicating whether the module is defective or not. In order to model the causes of defective modules, static measures obtained from source code, mainly size, combination, consistency, legacy, and complication measures are used to determine whether a module is defective or not. In [12], data mining techniques are used to search for rules that indicate modules with a high probability of being defective.

Other techniques include the use of support vector machines and SOA using expert models are used on different data repository to investigate the datasets denoting the project defects and bugs. The datasets are organized according to component size. They found improved prediction presentation in the subsets that included larger components. Based on these results, they developed some recommendations that experts can follow in their defect-prediction works. These recommendations can help to overcome the low forecast performance, reported by different datasets that are not compatible to the software component.

Though, these recommendations are not converted to a real software system that is integrated with other components to guarantee that software developers follow the guidelines and write code that minimize the probability of having defective modules. The absence of an established standard makes it hard, if not impossible, to compare approaches.

But, the results of this comparison do not indicate that a specific algorithm has significant better results in the competences of estimate. The abovementioned challenges can be summarized and there is a need for effective software tools that uses data from available software repositories to predict the defective modules of a new software project in order to enable the project managers and quality team to know where to invest their time in testing and fixing. As there is no decisive conclusion of the best data mining technique which solve the problem of predicting defective modules in software projects, these tools should apply and combine different data mining techniques and compare the results.

This research work contributes are worth thoughtfulness for providing a solution architecture that can enrich software development quality based on data in software repositories. Most of the above mentioned references adapted the software metrics into knowledge and rules that should be used by practiced development team members to enhance their software project code. Conversely, there is a lack of a software architecture that can be implemented and integrated with software system developers to use this extracted knowledge to warn the less experienced software development team members of potential defective modules and enable the project manager to assign more resources to these potential defective modules. The integration of the data mining models with bugs tracking databases and metrics extracted from software source code leads to have more accurate results.

Secondly, this research work provides a standard that provides an ensemble of data mining models in the defective modules prediction problem and compares the results. This standard has the same conditions including the same input dataset, the same percentage of the training dataset and the same feature selection

approach. This research work proposes a combined approach to get better likelihood results according to the values of correctness, recall, precision and F-measure.

The overall approach is called the ensemble approach as different methods are combined in a single process framework for generating appropriate results. The specific objectives of this work are to demonstrate a methodology for Extraction of Data and Analysis of Test cases which addresses the issues of finding the bug occurrence and processing the tests data in software development, improves the relevancy of the retrieved content and presents the final outcome in a user friendly manner. This paper is organized into five sections. The remainder of this paper is organized as follows: Section 2 discusses the overall methodology and the components of the ensemble approach used in this research. The operation of the system is described in Section 3. Section 4 describes the implementation and the results, the results of applying the data mining tools on several projects are presented. A detailed analysis and discussion are presented. Finally, Section 5 concludes the paper.

The proposed ensemble approach consists of four interlinked components for Extraction of knowledge from the test data sets. The approach consists of components for: a) data collection b) data pre-processor c) data post processor and d) results plotting. The overall system of the ensemble approach is shown in Figure 1.

In the data collection stage the test case data is collected from different sources with the help of data

Figure 1: Overall Process

collector and preprocess the data, the data processing is done with the help of different sub components that incorporated in the data preprocessing component and where related-X-n approach is used to the process the user given data. The related-X-n approach is a method by which the user data and its n related terms are sent to the data processor. The results of data pre-processor are sent to the post processing of data to find the defects occurring and the concentration of the occurrence of the defects using pattern identification tree based approach. Finally, using the metric evaluator the processed data is evaluated and the content selected by the user from the data is transformed by using a summarization – smoothening approach and the results shown to the users in a plotting view and the overall interaction is used in the interaction stage for implicit rule formation that helps refine the order of defective or bug analysis suggestion generation and result generation. The various components that implement the Ensemble approach are shown in Figure 2. There are four major components:

- Data Collector and pre-processor
- Data pattern identification
- Metrics to evaluate processed data
- Results plotting

1. DATA COLLECTOR AND PRE-PROCESSOR

In this components it is to mandatory to obtain data sets from different datasets of the software systems and also defects tracking systems. There are few steps to get the data sets and process from different systems:

a) To obtain the different changes in data log information from different version software means to obtain all the defects ids data which belong to certain version of software during software development.

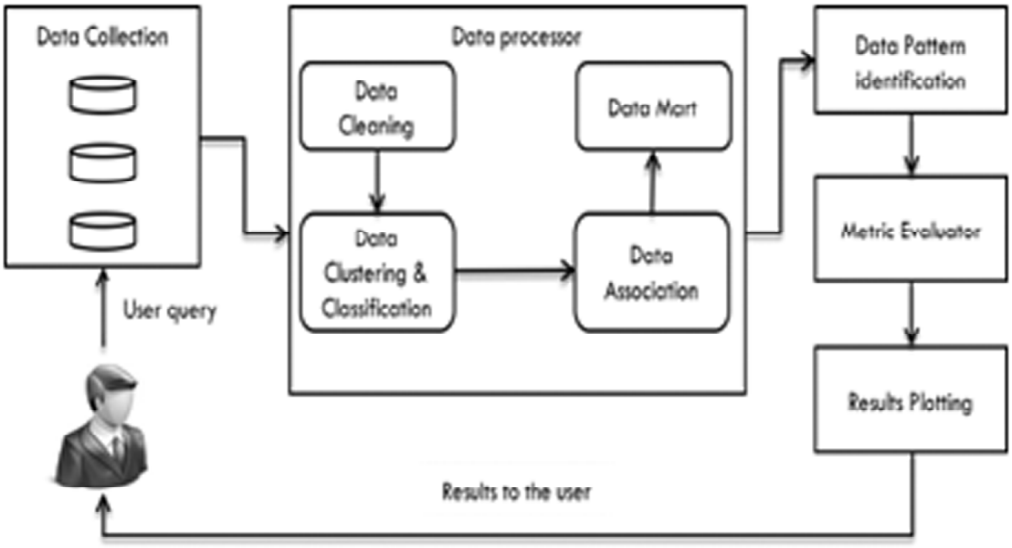

Figure 2: Components of Ensemble Approach

b) The changes in log data is related to the log file version from different software systems is something similar to bug 2345, and bug 2456, or follow up patch to bug 2567. The commit information like this contains the defect ID and related solutions. Where the user can only get the possible bug id number which contains the bug ID, the data is not so trustworthy.

c) The key problem is to improve the reliability of the results. From the acquired defect ID of bug, the version information in version control system on the specific location. Each bug report contains version information of this field, including a version from which the defect was narrated.

d) During the software development life cycle the information may change; in the end we may get more than one version of this data. We usually use the earliest version. If the defect was first discovered in the version similar to "version 1.01 releases" then system determined that the defect is a defect after the release, otherwise, we think it is a pre-release defect, which is discovered in the pre-release software development and testing process. Defects after the release of the software are the defects which found by software users after release.

e) Using the clustering algorithm the defects are clustered into groups based on the release and module which occurs. Where the association rules are used to categorize the bugs and this processed data will be stored in the data marts.

2. DATA PATTERN IDENTIFICATION AND RESULTS PLOTTING

In data pattern identification and results plotting process the processed data which is stored in the data marts are given to the identification process to analyse where the bugs are concentrated more in high amount and the results are plotted in different graphical representations to understand by the user. There are few steps to get the data sets and process from different systems:

a) For pattern identification the software metrics for files and packages which contain bugs are calculated and for this process, the Eclipse plugin Metrics 2.4.6 is used. This software metrics chosen in this research work is proposed by Sellers and Brian Henderson.

b) Where it is a set of object-oriented based software metrics; also include complexity metrics. The cyclomatic complexity metrics, calling other functions, nesting depth, the number of parameters, the number of domains, and number of methods are the types introduced in software metrics. So we need cluster analysis for these metrics, respectively, sum, max, average calculation.

c) The predicted results and the performance evaluation of the model is displayed to the user in graphical and tabulated format to evaluate the prediction result.

3. OPERATION OF THE PROPOSED FRAMEWORK

The defects classification prediction method proposed in this thesis contains four aspects, obtain the dataset, analysis dataset and pre-process, build prediction model based on different classifiers, and evaluate the performance of models. The steps are:

Step 1. Access the dataset from data sources

Step 2. Divide the dataset into few levels based on the dataset type

Step 3. Apply the data cleaning algorithm to remove the spaces and malicious data to obtain the information, which includes the pre-release, post-release defects number, id, file name, and the process id which contains the data.

Step 4. Cleaned data is processed through the cluster and clusters are formed based on the classification which is done based on the bug id and other information which obtain the statistic information.

Step 5. Processed data is stored in data marts

Step 6. For each cluster, using software metrics plugin the cluster will be analysed to calculate the software metrics

Step 7. For each cluster, it is calculated

Step 8. Repeat the data in step 4 and 5, the information is processed

Step 9. The predicted results and the performance evaluation of the model itself

From step 1 to 8 is the data acquire, data preprocess and data post process, list them here to show the relationship between them and the subsequent experiment. Here is some detail information for these 8 steps.

4. EXPERIMENT AND RESULTS

Table 1 shows, the analysis of different analysis arose from the necessity to know how many bugs and the concentrated area or modules in each version and the time dedicated to the accomplishment of each task. The results are depicted in a graph. This research work have realized the analysis to know the defected

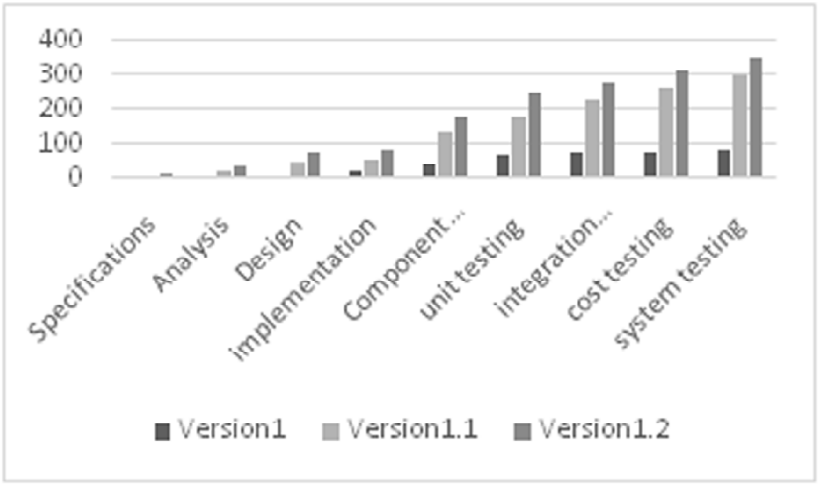

Figure 3: Number of bug defects in a project in different phases

modules and the average duration of each task based on the phases of the work. The data used to realize it, are the geometric averages of duration of the task. In figure 4.1 it shows the different defects that are occurred in same project in different versions in different stages of project development.

The Figure 4.1 shows a progressive increase of the bugs of different tasks from analysis phase until user testing phases; later, the duration descend to values similar to the starting phases of the development. In conclusion, of this research work could say that if the "evaluation" task and "analysis" task they do not last much then extends the duration of the" resolution" task. However later, the task of identifying bugs gets shorter at the same time as "evaluation" and "analysis" tasks extends their duration. More resolution appears in the beginning, and more analysis and evaluation appears after the Implementation phase, which is quite normal because they are testing, so they are really oriented to finding the defects. Whereas in the beginning after some review, they are fixing the problems at "resolution", but takes more time because they are not doing the "analysis". In the other hand, is quite normal to spend more time to fix the defects at the beginning of the project because the workers have to get used with the new software.

5. CONCLUSION

This research work has been focused on the database form the different versions of the software development testing process, which is in charge of solving the defects found during the development of a software process. The testing process belongs to a SDLC Process of an anonymous Indian company.

The proposed framework has a linear sequence of tasks executed since the data of defect is submitted until the process completes. That sequence of tasks is typified in the company, in the quality manual, with the purpose to be carried out for every new defect. This purpose is achieved because it was discovered that the various phases of different task, which is a task of that process, was approximately passed a 70% of cases.

In this system, a data mining ensemble approach is proposed. The key aspect of the system are methods for handling the data from different sources by the use of a related-X-n approach for data processing and a method for plotting the results. The results are encouraging. In future, the experimentation will focus on the performance of the system as per the different data combinations like from different versions and different phases of software development life cycle. In addition, the overhead of the system will be tested. A mechanism for processing the data is also proposed in this work. This will be expanded further. The work is general and can be applied for other software projects too. Hence, the emphasis can be on validating this aspect too.

REFERENCES

[1] Sunghun Kim, Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, and Shivkumar Shivaji. "Predicting Method Crashes with Bytecode Operations", In *Indian Software Engineering Conference*, pages 3-12, (2013).

[2] J. H Andrews, T. Menzies, and F. C.H Li. "Genetic Algorithms for Randomized Unit Testing", *IEEE Transactions on Software Engineering*, 37(1):80-94, (2011).

[3] J. Anvik, L. Hiew, and G.C. Murphy. "Who Should Fix this Bug", *In Proceedings of the 28th International Conference on Software Engineering*.Pages 361-370, (2006).

[4] Lamkan , S. Demeyer, E. Giger, and B. Goethals. "Predicting the Severity of a Reported Bug.",*In Proceedings of the 7th Working Conference on Mining Software Repositories*, pages 1-10. IEEE, (2010).

[5] Arcuri A and L Briand. "A Practical Guide for using Statistical Tests to Assess Randomized Algorithms in Software Engineering",*In Proceedings of the 33rd International Conference on Software Engineering*, pages 1-10, (2011).

[6] Djuradj Babich, Peter J. Clarke, James F. Power, and B. M. Golam Kibria. "Using a Class Abstraction Technique to Predict Faults in OO Classes: a Case Study Through Six Releases of the Eclipse JDT",*In Proceedings of the ACM Symposium on Applied Computing*, Pages 1419-1424, (2011).

[7] Al Bessey, Ken Block, Benjamin Chelf, Andy Chou, and Dawson R. Engler. "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World", Communications of the ACM, 53(2):66-75, (2010).

[8] Christian Bird, Adrian Bachmann, Eirik Aune, and Premkumar Devanbu. "Fair and Balanced?: Bias in Bug-Fix Datasets",*In Proceedings of the 7th joint meeting of the European SoftwareEngineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ESEC/FSE '09, pages 121-130, (2009).

[9] Lucia Lucia, David Lo, Lingxiao Jiang, and Aditya Budi. "Active Refinement of Clone Anomaly Reports",*In Proceedings of the 34th International Conference on Software Engineering*, (2012).

[10] E. Shihab, Z.M. Jiang, W.M. Ibrahim, B. Adams, and A.E. Hassan. "Understanding the Impact of Code and Process Metrics on Post-Release Defects: a Case Study on the Eclipse Project",*Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1-10, (2010).

[11] S. Shivaji, E.J. Whitehead Jr, R. Akella, and S. Kim. "Reducing Features to Improve Code Change Based Bug Prediction", IEEE Transactions on Software Engineering, (2012).

[12] Yi Wei, Carlo A Furia, Nikolay Kazmin, and Bertrand Meyer. "Inferring Better Contracts",*In Proceeding of the 33rd International Conference on Software Engineering*, pages 191-200, Waikiki, Honolulu, HI, USA, (2011).

[13] Shivkumar Hasmukhrai Trivedi, "Software Testing Techniques" ISSN: 2277 128X International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE) Vol.2, Issue 10, October (2012).

[14] Dinesh Mavaluru, R. Shriram and W. Aisha Banu, "Ensemble Approach for Cross Language Information Retrieval", *in Springer, Lecture Notes in Computer Science*, Vol.,2, pp. 274-286, H-Index -100, ISSN No: 0302-9743, (2012).