Predicting Bug Inducing Source Code Change Patterns

Ayazuddin Khan[1], Syed Nadeem Ahsan[2]
Faculty of Engineering, Science and Technology Main Campus, Iqra University, Karachi, Pakistan.
Email: ayaz0306@yahoo.com[1], sn_ahsan@yahoo.com[2]

Abstract – **A change in source code without the prior analysis of its impact may generate one or more defects. Fixing of such defects consumes maintenance time which ultimately increases the cost of software maintenance. Therefore, in the recent years, several research works have been done to develop techniques for the automatic impact analysis of changes in source code. In this paper, we propose to use Frequent Pattern Mining (FPM) technique of machine learning for the automatic impact analysis of those changes in source code which may induce bugs. Therefore, to find patterns associated with some specific types of software changes, we applied FPM's algorithms' Apriori and Predictive Apriori on the stored data of software changes of the following three Open-Source Software (OSS) projects: Mozilla, GNOME, and Eclipse. We grouped the data of software changes into two major categories: changes to meet bug fixing requirements and changes to meet requirements other than bug fixing. In the case of bug fixing requirements, we predict source files which are frequently changed together to fix any one of the following four types of bugs related to: memory (MEMORY), variables locking (LOCK), system (SYSTEM) and graphical user interface (UI). Our experimental results predict several interesting software change patterns which may induce bugs. The obtained bug inducing patterns have high confidence and accuracy value i.e., more than 90%.**

***Keywords*: Software Changes; Frequent Pattern Mining; Impact Analysis; Specific Types of Bugs; Transaction Data**

I. INTRODUCTION

Software maintenance i.e. changing in source code is typically supposed to be expensive, complicated and time consuming. One of the reasons for the same is its level of multifaceted nature for developers and software quality assurance (SQA) personnel to fully determine a program's structure that they did not consider or implement. Even when they evolve their own structure, it is, therefore, so straightforward to overlook the trivial detail of source code. [3, 6, 7]

Software change Impact Analysis is normally used to calculate the cost of a change in source code after that change has been made. However a more proactive set about values impact the analysis to predict the effects of the change before it is started. Predictive impact analysis can allow developers attempting to an outline, as opposed to just manage outcomes or progressively outstretching influence of changes in source code [12]. Change impact analysis is widely used to identify the potential consequences of a change in a source code. In other words analyse what else will likewise be required to change avoiding any undesired outcome from the change in the source code. This is one of the most significant areas in software maintenance as nonexistence of appropriate change impact analysis may amplify software maintenance time and cost extensively. Therefore, in the recent years some research works have been done to formulate models for the impact analysis of software changes. Generally, various techniques of machine learning have been used to formulate models for the impact analysis of software changes.

Following are our research hypotheses which are based on software evolutionary data collected during the development of software product:

H-1: Using software evolutionary data, we can predict the group of source files which are frequently changed together to meet certain requirements of software changes. These change requirements may be categorized into two major categories: (i) change requirement to fix bugs and (ii) change requirements other than bug fixing like code enhancement, adding new feature, increasing adoptability.

H-2: Moreover, we can predict the likelihood of changes in a file which may generate some specific types of bugs, which includes bugs related to memory, variables locking, system and graphical user interface.

To validate our hypotheses we performed an experiment by using software evolutionary data of following three OSS projects: Mozilla, GNOME, and Eclipse. We warehoused the data into a relational database. Finally, the stored data has been utilized to predict frequently change patterns and their association with some specific types of software change requests. Our key contribution in this research work is that we introduced a novel approach to predict source files which are frequently changed together to fix some specific types of bugs. For this purpose, frequent pattern mining has been used in an innovative manner.
This paper has been arranged as follow: In Section **II**, related work is examined. In Section **III**, background theory is portrayed. In Section **IV**, research work is talked about. In Section **V**, experimental setup is defined that how we performed our research work. In Section **VI**, results are discussed. In Section **VII**, the research work is concluded and open areas are identified.

II. RELATED WORK

Related work review demonstrates that other researchers have also worked in this area from various aspects. This review helped us to undertake our research work in a thoughtful and meaningful approach to accomplish the research objective.

The study of related research papers also improved our vision towards the use of machine learning techniques differently.

Fischer, Michael et al 2003 [2] utilized an approach to populate a release history database that consolidates version data with bug tracking data. They applied the method of simple database queries used to extract meaningful information from organized populated data.

Zimmermann, Thomas, et al 2004 [8] talked about pre-processing of CVS data to improve its quality and make it more suitable for loading into a relational database so that the analysis may be more beneficial. They basically explained four different pre-processing tasks which should be done to perform a fine-grained analysis from CVS archives. In our experimental work, we also performed various pre-processing activities to compile the extracted CVS data into meaningful database repository.

Zimmermann, Thomas, et al 2005 [1] exhibited data mining technique for software change impact analysis. They utilized a technique for extracting different information from large datasets and performed the predictive analysis of software changes. Source code maintenance regarding software change impact analysis is discussed using various rules and matrices. To accomplish this, they presented an eclipse plug-in tool i.e. Reengineering of Software Evolution (ROSE) which needs a version archive such as CVS and Bugzilla. The tool facilitates the programmers to predict likely changes, prevent errors due to incomplete changes and detect coupling undetectable by program analysis.

Sunghun Kim et al 2008 [12] presented a new technique for forecasting latent software bugs, called change classification which use a machine learning classifier. This approach predicts whether another software change is likely to be buggy or clean. Our research work is different from this approach as we associate with each transaction its type of bug for which it was changed.

Ahsan, Syed Nadeem et al 2010 [4] introduced two different approaches based on automatic text classification of Software Change Requests (SCRs). To attain this, they firstly applied Latent Semantic Indexing (LSI) to index the key terms of SCRs' summary. Then for classification they used two different techniques of machine learning i.e. single and multi-label classification.

Oliva, Gustavo A., et al 2013 [9] discussed a change impact analysis approach in which they depicted that large-scale workflow repositories are very complex to decide the effort of change and its subsequent impact. Repositories contain dependency of interconnected workflows which may produce undesirable behaviour and affect other parts of the system comprising the reliability of the repository. To overcome this issue, they presented a static dependency-centric approach for workflow repository management

Motahareh Bahrami Zanjani et al 2014 [10] presented an approach for impact analysis of an incoming software change request (SCR). This is based on a combination of interaction and commits histories. The software entities, files and methods, were interacted or changed in the resolution of earlier bug fixes.

Ayse Tosun Misirli et al 2016 [11] proposed a solution to predict software changes that are high impact fix-inducing changes (HIFCs). They debated that handling of all fix-inducing changes in the same way is not perfect because a small typo in a software change is easier to correct by a developer than some problematic issue while the earlier research work for the same has many advantages in terms of relatively accurate results. They suggested a solution to measure high impact fix-inducing changes that takes care for future bug fixing changes.

Our research work is an extension of the work done earlier by Thomas Zimmermann, et al 2005 [1] in which the developers were facilitated by an eclipse plug-in tool i.e. ROSE which suggests and predicts likely changes and prevents errors due to incomplete changes. We bring the novelty in our research work by extending the work to identify the relationship of co-changed files by their association with some specific types of bugs.

III. BACKGROUND THEORY

Maintenance of software is considered to be one of the most difficult activities in a Software Development Life Cycle (SDLC) because no software is supposed to be developed perfectly in a manner that does not require any change in future whereas they definitely need changes over a period of time [5]. Change impact may broadly be defined as the impact of any change in the existing source code of a program. Impact analysis may be done in two main ways. Firstly, before a change is made to predict change impact and cost estimation. Secondly, after a change is made to trace its effects [5].

A. Mining Software Repositories

Software repositories are rich storage of different source codes developed and maintained by developers through some version control systems like CVS, SVN, and Git etc. Software practitioner started mining these repositories and now they strongly believe that such repositories are very useful source for software maintenance. Mining Software Repositories (MSR) field evaluates the extended dataset available in software repositories to discover interesting and desired information about different software projects. Following is a brief overview for some technologies used in mining software repositories:

Version Control System is a methodology to maintain numerous concurrent developers working on the same software project. Developers get benefit from using such systems to keep copies of source code files.

Bug Tracking System is a process to store and maintain reported bugs against different software projects and keep updating their status as they are changed till they are either fixed or declared unfixed for some reason. **WEKA** is a commonly used open-source suite of machine learning software written in object oriented language Java.

B. Machine Learning Techniques

Machine Learning is a branch of artificial intelligence having various techniques. Algorithms used in machine learning are broadly categorized in two categories i.e. Supervised and Unsupervised Learning.

Association Rule Learning [14] is basically a subtype of unsupervised learning. They are simply IF/THEN statements which help to identify relationships between apparently unrelated data. An association rule has two parts, an antecedent [IF] and a consequent [THEN]. An antecedent is an item found in the data whereas a consequent is an item that is found in combination with the antecedent.

Every association rule has a support (s) and a confidence (c) where support is the percentage of transactions that contains two items X and Y whereas confidence checks how often item Y appears in transactions that contain X. Following are few techniques related to association rule are:

Frequent Pattern Mining (FPM) is one of the most common but important techniques for mining large data repositories. This helps in finding the patterns of frequent relevant data sets based on different rules. FPM study has considerably enhanced the scope of data analysis and will have definite impact on data mining methodologies in the long run.

Apriori Algorithm is a classic example for frequent item set mining and association rule learning over transactional database. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently in the database.

Predictive Apriori Algorithm is an improved version of Apriori Algorithm [15] which generates more confident rules and ranks the rules accordingly. One problem with general Apriori is that it will prepare general rule with high support and low confidence and very specific with high confidence and low support. Predictive Apriori evaluates the confidence of rules depend on their support. In this algorithm, support and confidence are combined into a single measure called predictive accuracy.

IV. RESEARCH APPROACH

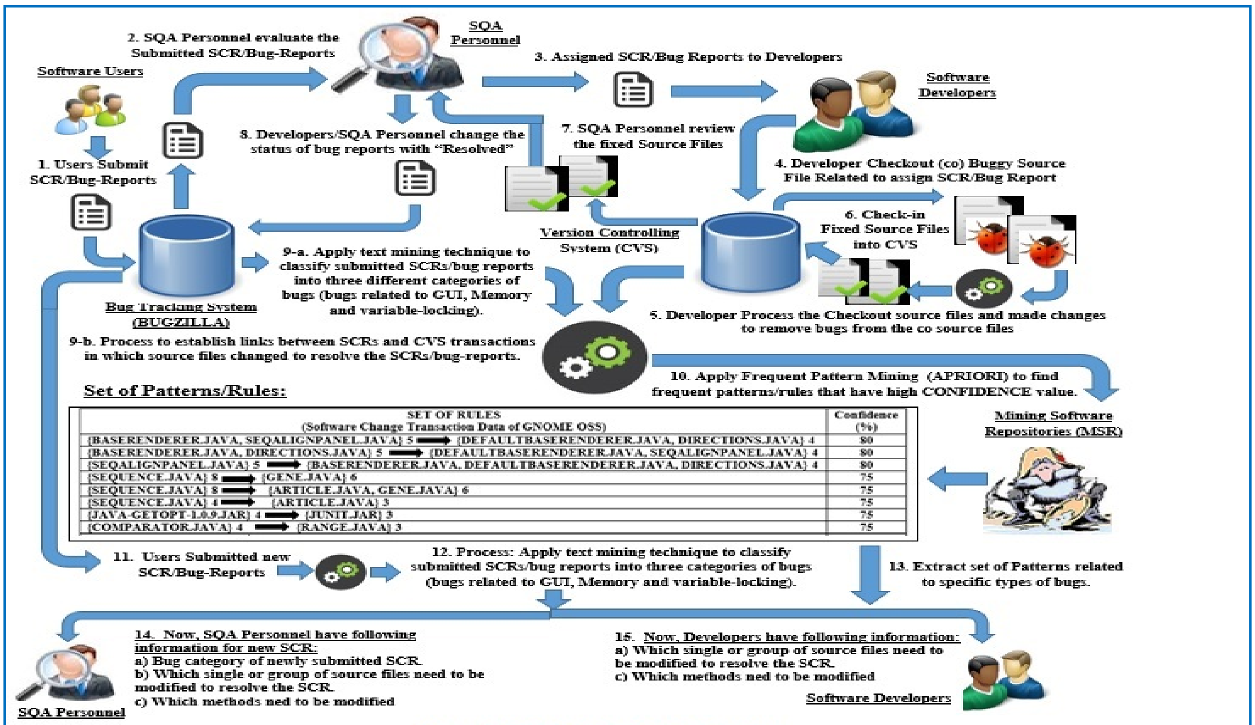
Our research work is based on our hypothesis i.e. frequent pattern mining techniques (Apriori and Predictive Apriori algorithms) of machine learning can be applied on software evolutionary data to predict group of source files or files' methods which are frequently changed together to fix some specific types of bugs. To validate our hypothesis, we propose an approach as described in the next paragraph.

In case of OSS, software change requests i.e. SCRs (also known as bug reports) are frequently submitted into a bug tracking system. After validation, the submitted SCRs are assigned to appropriate developers by Software Quality Assurance (SQA) personnel. The developers are then responsible to resolve the assigned SCRs. For this purpose, developer checkout relevant source file/files from version controlling system and made appropriate changes in the source file/files. Finally, the developer has to commit those changes into the CVS repository. The CVS commit process is also known as transaction. In such transactions developer may commit changes into a single or multiple source files to resolve the submitted SCR/bug-report. Now, it is expected that in the future if similar SCR/bug-report are submitted again, then the developer has to make changes in those source files in which he did changes last time to fix the similar type of SCR/bug-report. Therefore, if we have the previous data of changes in source codes and associated SCRs, then we can use that data and build machine learning based software engineering tool which can automatically generate different patterns/rules and support software developers/SQAs in engineering tool which can automatically generate different patterns/rules and support software developers/SQAs in making decision about which source files or methods need to be modified to resolve the submitted issue. Each patterns or rules can be evaluated on the basis of two metrics of association rules or patterns i.e. (1) Support and (2) Confidence. Since, our research objective is to predict those change patterns in which multiple source files are frequently changed together to fix specific type of bugs.

Therefore, we extracted the data of such transactions from CVS repository in which multiple source files have been changed and establish its link with associated bug reports using Bugzilla repository. Finally, we applied Apriori and Predictive Apriori algorithms to identify file level and method level frequent change patterns which are associated with the following four types of bugs related to: memory (MEMORY), variables locking (LOCK), system (SYSTEM) and graphical user interface (UI).The steps 1-to-15 depict our research approach in details. It is shown in the figure that there are total 15 steps. The steps 1-to-8 are used to extract data from Bugzilla and CVS repositories, whereas, the intermediate steps i.e. steps 9 and 10 are used to process the extracted data. In step 9a, we process SCR data and apply K-means clustering text mining technique to identify the type of SCRs. For this research work we only considered four types of bugs. If the submitted SCRs is related to bug fixing then it belongs to anyone of the above mentioned bug categories, otherwise the submitted bug reports is not related to bug fixing. In step 9b, we establish link between SCRs and associated changes which have been committed in CVS. Whereas, in step 10, we apply APRIORI algorithm to find change patterns which are frequently made by developer to fix some specific types of bugs. Finally, a set frequent change pattern are available for SQA personnel and developers to guide them that which source files or methods need to be modified to resolve the submitted SCRs. Steps 11-to-15 are final steps which shows that how frequent change patterns of source files / methods facilitate SQAs and developers to accomplish their tasks.

Below-mentioned Figure 1 also shows an illustration of our proposed research approach.

levels. As may be required there are various options to print the relevant reports as well.


SQA Personnel
2. SQA Personnel evaluate the Submitted SCR/Bug-Reports
Software Users
3. Assigned SCR/Bug Reports to Developers
Software Developers
1. Users Submit SCR/Bug-Reports
8. Developers/SQA Personnel change the status of bug reports with "Resolved"
7. SQA Personnel review the fixed Source Files
4. Developer Checkout (co) Buggy Source File Related to assign SCR/Bug Report
6. Check-in Fixed Source Files into CVS
Version Controlling System (CVS)
9-a. Apply text mining technique to classify submitted SCRs/bug reports into three different categories of bugs (bugs related to GUI, Memory and variable-locking).
Bug Tracking System (BUGZILLA)
5. Developer Process the Checkout source files and made changes to remove bugs from the co source files
9-b. Process to establish links between SCRs and CVS transactions in which source files changed to resolve the SCRs/bug-reports.
10. Apply Frequent Pattern Mining (APRIORI) to find frequent patterns/rules that have high CONFIDENCE value.
Set of Patterns/Rules:
Mining Software Repositories (MSR)
11. Users Submitted new SCR/Bug-Reports
12. Process: Apply text mining technique to classify submitted SCRs/bug reports into three categories of bugs (bugs related to GUI, Memory and variable-locking).
13. Extract set of Patterns related to specific types of bugs.
14. Now, SQA Personnel have following information for new SCR:
a) Bug category of newly submitted SCR.
b) Which single or group of source files need to be modified to resolve the SCR.
c) Which methods ned to be modified
SQA Personnel
15. Now, Developers have following information:
a) Which single or group of source files need to be modified to resolve the SCR.
c) Which methods ned to be modified
Software Developers

SET OF RULES (Software Change Transaction Data of GNOME OSS)	Confidence (%)
{BASERENDERER.JAVA, SEQALIGNPANEL.JAVA} 5 $\longrightarrow$ {DEFAULTBASERENDERER.JAVA, DIRECTIONS.JAVA} 4	80
{BASERENDERER.JAVA, DIRECTIONS.JAVA} 5 $\longrightarrow$ {DEFAULTBASERENDERER.JAVA, SEQALIGNPANEL.JAVA} 4	80
{SEQALIGNPANEL.JAVA} 5 $\longrightarrow$ {BASERENDERER.JAVA, DEFAULTBASERENDERER.JAVA, DIRECTIONS.JAVA} 4	80
{SEQUENCE.JAVA} 8 $\longrightarrow$ {GENE.JAVA} 6	75
{SEQUENCE.JAVA} 8 $\longrightarrow$ {ARTICLE.JAVA, GENE.JAVA} 6	75
{SEQUENCE.JAVA} 4 $\longrightarrow$ {ARTICLE.JAVA} 3	75
{JAVA-GETOPT-1.0.9.JAR} 4 $\longrightarrow$ {JUNIT.JAR} 3	75
{COMPARATOR.JAVA} 4 $\longrightarrow$ {RANGE.JAVA} 3	75

Figure 1: Overall System Process Diagram

V. EXPERIMENTAL SETUP

To perform an experiment, we downloaded software evolutionary data from the version controlling and bug tracking systems of the subsequent three OSS projects: Mozilla, GNOME, and Eclipse. The data are freely available and can be downloaded. Accordingly, we connected with the aforesaid OSS projects to download source, annotation, log files, and bug reports. It has been observed that in case of OSS projects, bug databases are not directly linked with a version control system. Therefore, there is no direct mapping between bug reports and the name of the fixed / updated source files. However, in recent years, a number of techniques have been developed to relate bug reports to fixes in source files. To establish a link between a bug report and the list of fixed source files, we used an approach similar to the approach of Fischer et al., 2003 [2]. In order to perform our experimental setup, we extracted unstructured data from the project repositories of Mozilla, GNOME, and Eclipse. After accomplishment of pre-processing steps, we stored the refined data into relational database for predicting bug inducing source code change patterns.

To present our research work discussed in this paper, a prototype front-end application has also been designed. To achieve this, open database connectivity was established from the front-end application to relational database and thereafter database views were created to use the same in the front-end application. Using this application a developer can connect with back-end tables and check for database repository review of processed data and impact analysis at different granularity

VI. RESULTS DISCUSSION

We extracted various CVS transactions based on single commit. This included 2457 transactions of Mozilla, 1047 of GNOME, and 3244 of Eclipse. We processed the downloaded data and applied frequent pattern mining techniques of machine learning to predict those patterns in which one or more files and files' methods frequently changed together to resolve specific types of bugs. We used Apriori algorithm to perform the next two experiments as defined in the following sections A and B.

A. FPM Approach at Granularity Level of Files

To perform inferential analysis, we extracted subset from CVS transactional data in which one or more files have been changed and established its link with SCR and then applied Apriori algorithm to obtain frequent pattern of transactions in which one or more files changed together. In this inferential analysis, there are extract of processed software change transactional data for three open-source projects: Mozilla, GNOME, and Eclipse predicting co-changed patterns at granularity level of files.

B. FPM Approach at Granularity Level of Methods

Now we moved one granularity level deep and checked in selected Java files that which method is changed. This was achieved through first generating annotation files from CVS checked-out project and identifying total revisions changed against each row. After fetching transactions for distinct files and methods and calculating changes against each method within transaction and changes against each method within overall database for onward calculating support and confidence for each distinct row. As defined in Experimental Setup section, we calculated support and confidence for each distinct row based on Transaction ID File Name and Method Name. This working is done through CVS, PL/SQL Queries, and Front-end Application.

GUI interface is used to retrieve any source code file and open the same in front-end application which is required to be changed and the developer wants to know its subsequent impact of the change to avoid any anonymous post change impact. This will not only facilitate the developer but it will also minimize software maintenance cost which is usually considered to be high, complex and time consuming. Once any source code fie is selected through front-end application, we may perform method level frequent pattern mining.

C. Impact Analysis for Specific Types of Bugs

This segment defines the methodology of the proposed technique that has not been used earlier. To perform impact analysis for specific types of bugs, relationship of files is determined by their association with some specific bug types. We worked to predict source files which are co-changed to fix any one of the subsequent four types of bugs
related to: MEMORY, LOCK, SYSTEM and UI. To achieve this, we created a table with bugs' related information and assigned every bug an associated bug class based on its associated semantic keywords. Semantic keywords here refer to some property of the changes such as occurrence of specific words like OVERFLOW, HALT, OPERATING and INTERFACE etc. that are present in CVS log files. This is required because to classify software changes, we do not have any known collection of written texts like UCI Repository of Machine Learning [13] which is widely used to evaluate the text classification.

Going forward, we proceed further in our innovative approach by performing FPM at granularity level of files along-with their aforementioned assigned bug class having specific types of bugs to determine co-change patterns associated with their respective types of bugs. Tables 1 and 2 show the comparative set of rules found after processing through Apriori and Predictive Apriori algorithms for one of the open-source projects i.e. Mozilla. This working clearly shows that as we go deeper with associated bug class, the confidence and accuracy for set of rules found are increasing and thus it can be revealed that the approach is more accurate as compared to traditional approaches for impact analysis of software changes which commonly use files, methods and classes etc. without their associated bug types.

Another set of selected rules show that if we focus the processed data on software changes associated with Buggy and Clean types then the confidence and accuracy for set of rules found are increasing. This working as shown in Tables 3 and 4 further support our research approach in getting the research objective done successfully.

Table 1: Comparative Software Change Impact Analysis - Traditional Approach

SET OF RULES - Software Change Transaction Data without any Associated Bug Class	Conf.	Acc.
{nsMacShellService.cpp nsWindowsShellService.cpp} 7 ➔ {nsGNOMEShellService.cpp} 5	0.71	0.66667
{WebContentConverter.js} 11 ➔ {addFeedReader.js} 7	0.64	0.61538
{nsMacShellService.cpp} 8 ➔ {nsWindowsShellService.cpp} 7	0.88	0.80006
{calGoogleSession.js} 10 ➔ {calGoogleUtils.js} 8	0.80	0.75000
{lightningTextCalendarConverter.js} 9 ➔ {calItipItem.js} 5	0.56	0.54545

Table 2: Comparative Software Change Impact Analysis - Proposed Approach

SET OF RULES - Software Change Transaction Data with Associated Bug Class (Bug Type)	Conf.	Acc.
{nsMacShellService.cpp nsWindowsShellService.cpp} 4 ➔ {nsGNOMEShellService.cpp CLASS=LOCK} 4	1.00	0.93897
{WebContentConverter.js} 6 ➔ {addFeedReader.js CLASS=UI} 5	0.83	0.75004
{nsMacShellService.cpp} 5 ➔ {nsWindowsShellService.cpp CLASS=LOCK} 4	0.80	0.75000
{calGoogleSession.js} 10 ➔ {calGoogleUtils.js CLASS=UI} 8	0.80	0.71432
{lightningTextCalendarConverter.js} 7 ➔ {calItipItem.js CLASS=UI} 5	0.71	0.66667

Table 3: Impact Analysis for Buggy Software Changes

SET OF RULES - Software Change Transaction Data for BUGGY Software Changes	Conf.	Acc.
{nsMacShellService.cpp nsWindowsShellService.cpp} 7 ➔ {CLASS=BUGGY} 7	1.00	1.00000
{WebContentConverter.js} 11 ➔ {CLASS=BUGGY} 9	0.82	0.76923
{nsMacShellService.cpp} 8 ➔ {CLASS=BUGGY 8}	1.00	1.00000
{calGoogleSession.js} 10 ➔ {CLASS=BUGGY} 10	1.00	1.00000
{lightningTextCalendarConverter.js} 9 ➔ {CLASS=BUGGY} 7	0.78	0.72727

Table 4: Impact Analysis for Clean Software Changes

SET OF RULES - Software Change Transaction Data for CLEAN Software Changes	Conf.	Acc.
{nsMorkHistoryImporter.cpp} 10 → {CLASS=CLEAN} 10	1.00	1.00000
{controller.js places.js} 7 → {CLASS=CLEAN} 7	1.00	1.00000
{nsNavHistoryResult.cpp} 55 → {CLASS=CLEAN} 46	0.84	0.82456
{sidebarOverlay.js} 6 → {CLASS=CLEAN} 5	0.83	0.75004
{nsBrowserApp.cpp} 9 → {CLASS=CLEAN} 6	0.67	0.63636

The numerical values next to Antecedents and Consequents in Tables 1-to-4 show their number of occurrence in the processed data. To further validate aforementioned tables, following is a graphical illustration for comparative software change impact analysis. Figures 2 and 3 show that as we move from traditional approach i.e. basic theories related to FPM [16] towards the proposed approach i.e. FPM with associated bug class, the relevant confidence value is being increased substantially.

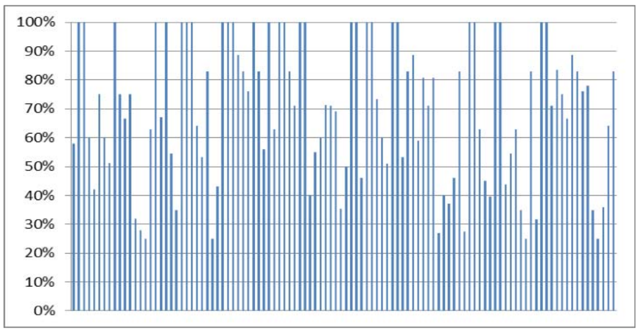


Figure 2: Traditional Approach

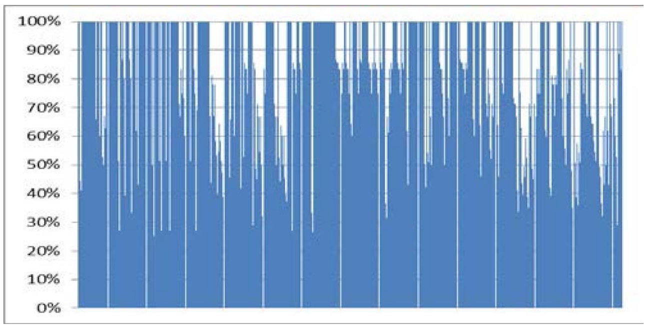


Figure 3: Proposed Approach

X-Axis = No. of Rules; Y-Axis = Confidence Value of Rules

VII. CONCLUSION AND FUTURE WORK

In the paper, we provided a study of software change impact and its related work to facilitate the developers in identifying the subsequent impact while making changes in the existing source-code files. Moreover, we extended the work to determine relationship of files through their association with some specific types of bugs. To achieve this we firstly classified the open-source data into various bug types and then compared the changed files based on classification of bugs.

As per our research hypotheses, machine learning technique has been very useful to complete our research work in the desired manner.

We believe that there has been done earlier research work on this area. However, there is a definite scope of more sophisticated and automated work particularly with reference to Frequent Pattern Mining to get relationship amongst software changes and predict more accurate and probabilistic impact of any software change being done by developers. There are also prospects of using Predictive Apriori and FP-Growth algorithms extensively to enhance the diversity of the future work in this area.

REFERENCES

[1] Zimmermann T, Zeller A, Weissgerber P, Diehl S. Mining version histories to guide software changes. IEEE Transactions on Software Engineering. 2005 Jun;31(6):429-45.

[2] Fischer M, Pinzger M, Gall H. Populating a release history database from version control and bug tracking systems. In Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on 2003 Sep 22 (pp. 23-32). IEEE.

[3] Ibrahim S, Idris NB, Munro M, Deraman A. Integrating Software Traceability for Change Impact Analysis. Int. Arab J. Inf. Technol.. 2005 Oct;2(4):301-8.

[4] Ahsan SN, Wotawa F. Impact analysis of SCRs using single and multi-label machine learning classification. InProceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement 2010 Sep 16 (p. 51). ACM.

[5] Li B, Sun X, Leung H, Zhang S. A survey of code-based change impact analysis techniques. Software Testing, Verification and Reliability. 2013;23(8):613-46.

[6] Law J, Rothermel G. Whole program path-based dynamic impact analysis. InSoftware Engineering, 2003. Proceedings. 25th International Conference on 2003 May 3 (pp. 308-318). IEEE.

[7] Ajila S. Software maintenance: an approach to impact analysis of objects change. Software: Practice and Experience. 1995 Oct 1;25(10):1155-81.

[8] Zimmermann T, Weißgerber P. Preprocessing CVS data for fine-grained analysis. InProceedings of the First International Workshop on Mining Software Repositories 2004 May 25 (pp. 2-6)

[9] Oliva GA, Gerosa MA, Milojicic D, Smith V. A change impact analysis approach for workflow repository management. InWeb Services (ICWS), 2013 IEEE 20th International Conference on 2013 Jun 28 (pp. 308-315).

[10] Zanjani MB, Swartzendruber G, Kagdi H. Impact analysis of change requests on source code based on interaction and

commit histories. InProceedings of the 11th Working Conference on Mining Software Repositories 2014 (pp. 162-171). ACM.

[11] Misirli AT, Shihab E, Kamei Y. Studying high impact fix-inducing changes. Empirical Software Engineering. 2016 Apr 1;21(2):605-41.

[12] Kim S, Whitehead Jr EJ, Zhang Y. Classifying software changes: Clean or buggy?. IEEE Transactions on Software Engg. 2008;34(2):181-96.

[13] Blake C, Merz CJ. 1998 {UCI} Repository of machine learning databases.

[14] BAher S, LMR J L. A comparative study of association rule algorithms for course recommender system in e-learning. International Journal of Computer Applications. 2012;39(1):48-52.

[15] Patil BM, Joshi RC, Toshniwal D. Classification of type-2 diabetic patients by using Apriori and predictive Apriori. International Journal of Computational Vision and Robotics. 2011 Jan 1;2(3):254-65.

[16] Shweta M, Garg DK. Mining Efficient Association Rules Through Apriori Algorithm Using Attributes and Comparative Analysis of Various Association Rule Algorithms. International Journal of Advanced Research in Computer Science and Software Engineering. 2013 Jun;3(6):306-12.