International Journal of Latest Trends in Engineering and Technology
Vol.(7)Issue(3), pp. 088-097
DOI: http://dx.doi.org/10.21172/1.73.512
e-ISSN:2278-621X

Investigation of Source Code Mining using Novel Code Mining Parameter Matrix: Recent State of Art

K Venkata Ramana[1] and Dr K Venu Gopala Rao[2]

Abstract— With the recent growing demand for higher accuracy with better control is every industry and research domains, the desired accuracy without compromising on the quality assurance policies can be achieved through automation. The automation is solely depended on software programs or codes to replacement the manual methods. Thus the codes desires to be quality assured. However the testing for software code quality is a tedious and time consuming process. Hence to shorten the process various data mining techniques and tools are been deployed to make timely prediction of the specific pattern of the defects in the code and proceed with the highly ranked faults. This paper evaluates the various data mining methods and tools for predicting software defects like violation of standard programming rules, faulty code segment and wrong programming API usage.

Keywords— Source Code Mining, Rule Violation, Predictive Analysis of Source Code. API Usage in Code

I. INTRODUCTION

The software projects without a proper detection technique for defects in developed modules may lead to software full of problems and not generating desired output mentioned in the required specification by customer. The requirements are generated during the mutual discussion between client and the development companies in the initial phases. The software development companies are always liable to deliver the software with agreed performance. Hence the software companies put lot of efforts in detecting and resolving the defects in the software. However the detecting and fixing process of detects is time consuming process and ignoring the defects may lead to malfunctions ranging from losing a small penny to million dollar loss. Software development companies deal with defects which are known and predictable and sometimes unknown and unpredictable defects. The known defects can be deal with pre-planned development strategies and are generally less time consuming. The known defects will not disturb the cost and time estimations for the project. However the unknown defects are unpredictable, hence the resolution of those defects also cannot be pre-defined. Hence the development industries keep huge efforts to deploy multiple prediction techniques to detect unknown defects for the next modules from the existing defect matrix generated during the development phases. The early documents have demonstrated the use of software defect matrix to demonstrate the time complexity, memory requirements and development cost in terms of time to market. An in depth calculation steps are to be executed to determine the number of defects in the software module or the complete program. However the recent researches have demonstrated the use of software defect matrix to form the guidelines for defect detection. The parallel researches have also demonstrated good classification techniques for defects matrices to focus on one specific objective and ignore the rest of the matrices for the same objective. The rest of the matrices can also

[1] Department of Computer Science & Engineering, Bhoj Reddy Engineering College for Women, Hyderabad, Telangana, India
[2] Department of Computer Science & Engineering, G.Narayanamma Institute of Technology and Science, Hyderabad, Telangana

be mapped to other objectives present in the process. Here it is to be understood that the objectives are concerned to quality assurance and defect resolution only.

In the high quality software demand driven world, the timely delivery of the project is also a major quality demand. Thus various research outcomes of data mining techniques [1] are been deployed to detect software quality and development productivity. The automatic detection of possible or existing defects in the code detects the defects and marks those defects in the code. Henceforth, the developers can generate the test cases to check for those patterned defects and remove multiple issues from the code easily and quickly.

The majority of the software defects relates to the following faults during the development:

a) Violation of software development rules
b) Violation of source code cloning techniques and
c) Violation or faulty use of software APIs

Firstly, detects violation of software development rules or commonly known as rule mining techniques. The violation of the software development rules can lead to the major technical and legal issues and the set of rules are generally unique for each specific development area or company, making it crucial and major threat. The researchers have demonstrated the successful use of data mining algorithms to find the rules from existing projects to improve the quality of the software development in the new project. Those demonstrated techniques help in finding the violation of source code rules in the new project and reduces the manual efforts to detect the rule violations.

Secondly, cloning of the source code from other existing project is a common practice to reduce the risk of new defects. Nevertheless, this increases the chance of higher maintainability. The requirement modifications once implemented in any of the part, the same needs to be reflected in all cloned parts. Many of the organization developed a strategy of using GitHub for code replication, which intern increase the change of software code piracy and security.

Thirdly, the application development industry demands external interfaces with third party applications. Most of the cases the external interfacings with the software codes are done by APIs. The inclusions of external APIs are not risk free with the consideration of the vulnerable information available within the software code. Hence, the developers must follow standard security measures to make the integration secured.

A number of researches made significant outcomes to address these above mentioned issues. Conversely, it is also to be noted that none of the existing research outcomes address the solutions to all existing problems through data mining techniques or algorithms or tools. Thus in order to propose a new approach or tools or algorithm it is important to analyse the existing tools and techniques with the quality of measurements and predictions using data mining techniques.

Thus this paper proposes two major outcomes as defining a quality matrix for software source code mining and analyse most of the code mining techniques and tools.

Henceforth, the rest of the paper is organized as in section II, the comprehensive literature survey is been presented, in section III this paper analyses the software defect matrixes, in section IV this paper purposes the software code mining parameters or quality matrix, in section V the comparative study of the source code mining techniques and tools are been analysed, in Section VI the goals for the further research directions are proposed and in section VII the conclusion is presented.

II. OUTCOMES OF THE PARALLEL RESEARCHES

Multiple parallel outcomes are been made to predict the software defects concerning the quality for software code development. The defects which affect the software development quality policies are listed in the introduction of this work. Thus this work analyses the outcomes of the parallel researches.

Firstly, in case of rule mining techniques to detect the rule extension defects a wide range of researches are been made. The detection includes rule extension, private edit violation and most prominently variable – data type violations. The contributions as Engler et al [2] and PR-Miner [3] for their function rule mining techniques have created the land marks in the research. Also the contribution of Chronicler [4] and Chang et al [5] for conditional rule mining is also proven to be

important. Finally the approached MUVI [6] for variable-pairing is also considered to be a motivation for further research.

Secondly, the cloning of source code leading to high maintainability for up gradations is another area for focus. The efforts by CCFinder [7] and Dup [8] by using tokenization are notable. Another approach based on abstract syntax tree [9] PGDs [10], CP-Miner [11] demonstrates the branching of source code and replication detection.

Lastly, the efforts by many other parallel researchers for API usage fault detection are also to be considered. The described standard pattern for API inclusion is demonstrated by many researchers [12 – 17] which also proven to be successful. The work of Michail et al [14] describes the use of item and association based rule reuse techniques. The work of Sahavechaphan and Claypool [15] developed, a context sensitive code assistant tool XSnippet for helping the developer for equivalence measurement of the source codes.

In the next section, this paper describes and analyse the software matrixes.

III. SOFTWARE MATRICES

The software matrices are created for various purposes ranging from describing the characteristics of the software product to information related to up-gradation to staff information deployed into the project. Software matrices are classified into major three categories as Product Related Matrix, Process Related Matrix and finally the Project Related Matrix [16]. In this work we understand different types and subtypes of these matrices:

A. *Product Related Matrix*

The product related matrices focuses on the product quality and the level of customer satisfactions. The matrices are designed to keep the information related to number of defects occurred in the software and level of customer satisfaction. The predicted information about how long the software will continue executing before failure is also stored in the matrices. A total of four sub-classifications are available for use under Product Related Matrix for various different purposes. We understand their purpose and use here [17]:

1) Matrix for Defect Density: The matrix for software defect density identification is majorly used for storing information related to number of defects in each software development modules. The process of calculation is as following, considering the density for defect for any module is d_{m1} defined as:

$$d_{m1} = \frac{\sum_{i=1}^{n} d_i}{n}, \qquad \text{…. Eq 1}$$

Where d_i denotes each defect in the module and n denotes the number of total number of lines in the software module. Based on the rate of defects the software product and the development companies are assured with the Capability Maturity Model CMM. The calculation for Capability Maturity Model is demonstrated in Table 1.

TABLE I: CAPABILITY MATURITY MODEL MEASUREMENT

Defect Rate	CMM Level can be achieved
Below 0.05	CMM 5
Above 0.05 and Below 0.14	CMM 4
Above 0.14 and Below 0.27	CMM 3

Above 0.27 and Below 0.44	CMM 2
Above 0.44 and Below 0.75	CMM 1

2) Matrix for Customer Feedback: The software matrix for customer feedback contains the information related to the reports reported by the customers. This matrix helps to understand and predict the amount of present and future defects to be addressed for each product or customer. The calculation for the problems can be calculated as followings, considering the number of problems as $P_{U/M}$,

$$P_{U/M} = \frac{\sum_{m=1}^{l} DP_m + \sum_{m=1}^{l} NDP_m}{l}, \qquad \text{…. Eq 2}$$

Where DP_m and NDP_m denote the number of defect problems & number of non-defect problems per months respectively and l denotes the number of months under license period.

3) Matrix for Customer Satisfaction: The matrix for customer satisfaction is based on the feedback data generated during the complete process of software development. The factors related to customer satisfaction are reliability of the software, responsive nature of the software, quality assurance of the software, applicable empathy of the software and tangibility of the software. The software development companies assign weightage for each factor considering the organizational and functional goals for software and each module inside that software [Table – 2].

TABLE II: SATISFACTION WEIGHTAGE MAPPING

Factors for Satisfaction	**Assigned Weightage**	**Weightage for each Modules**			
		M 1	**M 2**	**M 3**	**M 4**
Software Reliability	4	3	4	4	5
Responsive Nature	3	4	2	3	3
Quality Assurance	5	4	5	5	4
Software Empathy	4	4	4	4	4
Tangible Nature	3	3	2	2	3

Hence forth the feedback from the customer is been taken on the pre-decided questioner and then mapped to the feedback weightage. The final result of this process is the overall satisfaction rate or score.

B. Process Related Matrix

The main objective for software development to develop software which is satisfying all customer needs and in parallel it is also important to improve the software development process to achieve higher satisfaction rate during the further development tasks [18]. The software matrix for process contains information related to multiple factors concerning about the process of development in the organization. The following sub-categories are used for specific purposes:

1) Machine Testing – Defect Density Matrix: The defect density is measured during the system testing of the system. The system testing is performed manually or automatically in a simulated real time environment. The information stored in the defect density matrix is a nearly corrected correlation of the defects in the real time. Unless the assumptions of the

production system are massively incorrect, the defect density matrix can generate a good prediction.

2) Machine Testing – Defect Pattern Matrix: The total defects detected to be rectified are the overall scenario for the system. This might not be sufficient to predict the number of probable upcoming defects in the under development modules. Hence another matrix plays a role for prediction is Defect Pattern Matrix. This matrix denotes and helps in proper prediction of the defects which may be faced by the development team in under development modules for the same software product.

3) Defect Resolving Matrix: The detection of the defects is the half way task completion for the development team. The further task is to correct or resolve the defects. Another matrix called Defect Resolving or Removal matrix keeps track of the number of defects detected and resolved in the system. This helps in the identification of testing and resolution team efficiencies.

4) Effectiveness Matrix during Defect Removal: The input from the Defect Resolving Matrix is processed to generate the efficiency of Effectiveness Matrix. The effectiveness for each developed module can be calculated as following, considering DR_m is the efficiency for the module "m" as

$$DR_m = \frac{NDR_m}{TD_m} X100\%, \qquad \text{…. Eq 3}$$

Where NDR_m the number of defects is resolved and TD_m is the total number of defects detected in the module.

C. *Maintenance Related Matrix:*

After the completion of the development cycle, the software product is delivered to the client. Thereafter the maintenance process for the software starts including patch releases and corrective measured based on the customer feedback. Here the information is collected in multiple matrices to improve the defect resolution and customer satisfaction.

1) Matrix for Backlog Index: The backlog index denotes the ratio for number of defects resolved and number of defects reported. The formula for calculating the Backlog Index, B_I as following:

$$B_I = \frac{DR_m}{DD_m}, \qquad \text{…. Eq 4}$$

Where DR_m and DD_m are considered as Defects Resolved and Defects Detected in a month respectively.

2) Matrix for Fix Quality: Another matrix called Fix Quality matrix plays major role in defining the quality of the developed software delivered to the customer. The Fix Quality matrix denotes the ratio of number of total number of defects detected and resolved.

In the next section of this paper, the proposed code mining parameter matrix is proposed.

IV. PROPOSED SOURCE CODE MINING MATRIX

This work proposed a set of parameters to form the Code Mining Matrix. This matrix will help in understanding and proposing further improvements. The matrix consolidates the proposed parameters for rule based mining; code cloning and API pattern [Table – 3].

TABLE III PROPOSED SOURCE CODE MINING MATRIX

Mining Type	Matrix Parameters and Usage		
	Parameter Name	**Details**	**Expected Analysis**
Rule – Mining	Functions rules	The inference of the functions	Identify the function pairs that is been inferred
	Variable & Data Types	Variable Correlation Rules	Item Set Mining
	Dependence Graph	Graph based Conditional Rule Mining	Graph based Mining Analysis
	Global Class and Structural Variables	Variable Correlation Information	Variable Correlation Analysis
Code Cloning	Line_Seq	Sequence of Lines	Suffix Tree based analysis
	Token_Seq	Sequence of Tokens	Suffix Tree based analysis
	Statements	The cloning of the source code	Cloning Analysis
API Usage	API signature	API method signature	Analysis of Signature
	Class & Package	Sequence graph of the methods	Analysis of API calls in sequence
	Objects	List of Object creations	Analysis of Remote method invocations

In the next section this work lists the short comings of the existing code mining tools.

V. ANALYSIS OF CODE MINING TOOLS

A lot of automation tools are been introduced for code mining as an outcome of continuous research. However the tools individually are not sufficient to deliver the total analysis as per the proposed software mining analysis. Hence to understand the short coming this work analyses the short comings in the existing available tools [Table – 4].

TABLE IV ANALYSIS OF THE EXISTING CODE MINING TOOLS

Tool Name	Features and Short Comings of the Existing Tools	
	Feature	**Short Comings**
Static Analyser	Statistical analysis	Fixed rule templates, only identify pair wise programming rules
PR-Miner	Item-set mining	Does not consider inter-procedural analysis, data flow and control relationship
CHRONIC LER	Frequent subsequence mining	Does not take account of data flow or data dependence
Framework	Frequent itemset and subgraph mining algorithm	Require manual inspection for valid rules that may miss some instances of rules during inspection.
MUVI	Frequent itemset Mining	Only handled variable access directly by caller functions
Dup	Suffix tree based matching	Does not detect clone code portions having different syntax but similar meaning.
CC-Finder	Token comparison Suffix tree based matching	Does not detect changes such as statement reordering, insertion and

		control replacement.
CP-Miner	Frequent subsequence & tokenization	Same syntax but different semantic are detected as copy paste segments
Clone-Detection	Frequent item set mining	It does not detect complicated changes i.e. statement reordering, insertion and control replacement.
XSnippet	Graph mining	XSnippet is limited to the queries of a specific set of frameworks or libraries.
MAPO	Frequent sequence mining	It does not synthesized code fragments from mined frequent can be directly inserted into developers' code.
ParseWeb	Clustering	It only suggests the frequent MISs and code samples cannot directly generate compliable code.

The above carried out analysis will help to define the new technique and tools for the complete analysis based on the completeness of the proposed matrix.

VI. GOALS OF THE FURTHER RESEARCH

The further research direction clearly indicates enhancement in the code mining technique and implementation of a consolidated report generation tool as a final outcome of the research.

Code Mining based on Data Mining is a widely adopted technique and the data mining techniques are very useful for generating information for report generation and prediction processes. Hence the existing works clearly demonstrate the following data mining techniques to be adopted and evaluated:

- **Software Classification Modelling**: In the parallel researches the outcome of quality classification with the help of the existing dataset of software matrices is been demonstrated. However the approach used one software project matrices generated during the development process. To make the classification more robust, the researchers tried incorporating multiple other software project data. However the dataset from different projects are not compatible with the initial dataset. Hence a manual time consuming approach is been carried out to normalize the data. The work of Yi Liu et al. is to be considered as a bench mark for this model of work.

- **Association Based Rule Mining Method**: Use of Data Mining rules for establishing the correlation and prediction of software defects from the software matrices are also been proposed in parallel researches. The research conclusions are been applied to multiple project data for more efficient detection of software defects. The novel approach proposed by Song et al. is a notable mark in this direction of research and the approach proposed by them is also been compared with the PART or C4.5 algorithms to demonstrate the improvement. However this is to be understood that, generalizing the algorithm for over 150 projects is not a simple task and the approach can focus rather on normalization of the data.

- **Software Defect Prediction using Classifiers**: In other parallel tracks or researches also demonstrates the use of 22 most popular algorithms for data classification for defect prediction. The outcome of the research demonstrates that the algorithms demand the initial dataset to be in multiple dissimilar formats to be analysed. However the clarification algorithms demonstrate the same efficiency in detecting the defects. The work for Lessmann et al. is a benchmark for the comparative study.

Thus the following issues are to be addressed:

a) Implementation of novel framework for rule mining techniques and tools reducing Identify the function pairs that is been inferred, Item Set Mining, Graph based Mining Analysis and Variable Correlation Analysis
b) Implementation of novel framework for detecting source code cloning featuring Suffix Tree based analysis and Cloning Analysis
c) Implementation of novel framework for detecting faulty use of software APIs demonstrating Analysis of Signature, Analysis of API calls in sequence and Analysis of Remote method invocations

VII. CONCLUSION

This work comprehensively analyses of three different and most popular types of code mining techniques with the light of existing code mining tools. Rule mining techniques, source code cloning and framework for detecting faulty use of software APIs are been analysed. This work also analyses the strengths and shortcomings of the tools and technologies. The work proposes a complete and inclusive software matrix for performing a detail code mining and record the analysis for further prediction. Also the outcome of this work includes a complete road map for the further research directions with most desired features.

In the overall future direction of this work is a widespread framework for code mining.

REFERENCES

[1] A. Hassan, and T. Xie, "Mining software engineering data," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering- Volume 2, 2010, pp. 503-504.
[2] D. Engler, D. Chen, S. Hallem et al., "Bugs as deviant behaviour: A general approach to inferring errors in systems code," ACM SIGOPS Operating Systems Review, vol. 35, no. 5, pp. 57-72, 2001.
[3] Z. Li, and Y. Zhou, "PR-Miner: Automatically extracting implicit programming rules and detecting violations in large software code," in Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, 2005, pp. 306-315.
[4] M. Ramanathan, A. Grama, and S. Jagannathan, "Path-sensitive inference of function precedence protocols," in 29th International Conference on Software Engineering (ICSE 2007), 2007, pp. 240-250.
[5] R. Chang, A. Podgurski, and J. Yang, "Finding what's not there: a new approach to revealing neglected conditions in software," in Proceedings of the 2007 international symposium on Software testing and analysis, 2007, pp. 163-173.
[6] S. Lu, S. Park, C. Hu et al., "MUVI: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs," ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 103-116, 2007.
[7] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," IEEE Transactions on Software Engineering, pp. 654-670, 2002.
[8] B. Baker, "On finding duplication and nearduplication in large software systems," in Proc.Second IEEE Working Conf. Reverse Eng., 1995, pp. 86-95.
[9] V. Wahler, D. Seipel, J. Wolff et al., "Clone detection in source code by frequent itemset techniques," in Fourth IEEE International Workshop on Source Code Analysis and Manipulation, 2004, pp. 128-135.
[10] W. Qu, Y. Jia, and M. Jiang, "Pattern mining of cloned codes in software systems," Information Sciences, 2010, 2010.
[11] Z. Li, S. Lu, S. Myagmar et al., "CP-Miner: A tool for finding copy-paste and related bugs in operating system code," in Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation-Volume 6, 2004, pp. 20.

[12] M. Acharya, T. Xie, J. Pei et al., "Mining API patterns as partial orders from source code: from usage scenarios to specifications," in Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2007, pp. 25-34.
[13] D. Mandelin, L. Xu, R. Bodík et al., "Jungloid mining: helping to navigate the API jungle," ACM SIGPLAN Notices, vol. 40, no. 6, pp. 48-61, 2005.
[14] A. Michail, "Data mining library reuse patterns using generalized association rules," in Proceedings of 22nd International Conference on Software Engineering (ICSE'00), Limerick, Ireland, 2000, pp.167-176.
[15] N. Sahavechaphan, and K. Claypool, "XSnippet:mining for sample code," ACM SIGPLAN Notices, vol. 41, no. 10, pp. 413-430, 2006.
[16] C. Catal titled "Software fault prediction: A literature review and current trends published at Expert Systems with Applications Elsevier on 2011.
[17] A. G. Koru and H. Liu titled "Building Defect Prediction Models in Practice" published at Software IEEE Transactions on 2005.
[18] M. Mertic, M. Lenic, G. Stiglic and P. Kokol titled "Estimating Software Quality with Advanced Data Mining Techniques" published at Software Engineering Advances International Conference on 2006 at Tahiti.

About the Authors:

Mr.K Venkata Ramana was born in Guntur, Andhra Pradesh, in 1978. He received the M.Tech. degree in Computer Science & Engineering from JNT University, Hyderabad in 2010. He is currently pursuing Ph.D. degree in Computer Science & Engineering from JNT University, Hyderabad.He has 15 years of teaching experience. From 2006, he is working as an Associate Professor in the Department of Computer Science & Engineering, Bhoj Reddy Engineering College for Women, Hyderabad, Telangana, India. From 2001 to 2005, he worked as an Assistant Professor in the Department of Computer Applications, St. Johns Institute of Science & Technology, Hyderabad, Telangana, India. His research areas of interests are Data Mining, Machine Learning, Software Engineering, with an emphasis on mining software engineering data and software verification.

Dr K Venu Gopala Rao was born in Vijayawada, Andhra Pradesh, in 1963. He received the B.Tech. degree in Electronics and Communication Engineering from JNT University, Hyderabad in 1985, the M.Tech. degree in Computer Science and Engineering from the JNT University, Hyderabad in 1997 and Ph.D. degree in the area of Computer Science & Engineering from Osmania University, Hyderabad in 2008. From 2006, he is working as Professor, Department of Computer Science and Engineering at G.Narayanamma institute of technology and science (GNITS) Shaikpet, Hyderabad. From 2002 to 2006, he was an Associate Professor, in the Department of Computer Science and Engineering at G.Narayanamma Institute of Technology & Science Shaikpet, Hyderabad. From 1999 to 2002, he was an Associate Professor in the Department of Computer Science and Engineering at Koneru Lakshmaiah College of Engineering, Vijayawada, Greenfields, Vadeesvaram, Guntur. From 1997 to 99, he was an Assistant Professor in the Department of Computer Science and Engineering at VR Siddhartha Engineering college, Vijayawada. From 1992 to 1995, he was an Assistant Professor, Department of computer science & Engineering at JNTUCE, Hyderabad (1992-1995). From 1989 to 1992, he was a Technical Officer in CSG Group at ECIL Hyderabad. From 1988 to 1989 he was a Quality control engineer at Ashok Leyland, Hyderabad (1988-1989). From 1985 to 1988, he was an Engineer (Maintenance) at Radiant Cables Ltd, Hyderabad (1985-1988). His research interests include Network Security, data mining, statistical methods and their applications to software engineering.