IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 6, Ver. V (Nov – Dec. 2015), PP 25-30
www.iosrjournals.org

A Review on Software Fault Detection and Prevention Mechanism in Software Development Activities

B.Dhanalaxmi[1], Dr.G.Apparao Naidu[2], Dr.K.Anuradha[3]
[1]*Associate Professor, Institute of Aeronautical Engineering, IT Dept,Hyderabad,TS,India*
[2]*Professor , J.B.Institute of Engineering & Technology, CSE Dept, Hyderabad,TS,India*
[3]*Professor & HOD, GRIET, CSE Dept, Hyderabad,TS,India*

Abstract: *The need of distributed and complex commercial applications in enterprise demands error free and quality application systems. This makes it extremely important in software development to develop quality and fault free software. It is also very important to design reliable and easy to maintain as it involves a lot of human efforts, cost and time during software life cycle. A software development process performs various activities to minimize the faults such as fault prediction, detection, prevention and correction. This paper presents a survey on current practices for software fault detection and prevention mechanisms in the software development. It also discusses the advantages and limitations of these mechanisms which relates to the quality product development and maintenance.*

Keywords: *Software development, Fault Detection, Fault Prevention, Software faults*

I. Introduction

The software is a single entity which has established a strong impact on all the domain software which includes education, defence, medical, scientific, transportation, telecommunications and others. The activities of this domain always demands for high quality software for their accurate service need [1], [2], [3]. Software quality means to be an error-free product, which will be competent to produce predictable results and able to deliver within the constraints of time and cost. Therefore, a systematic approach for developing high quality software is increased in the competitiveness in today's business world, technology advances, the complexity of the hardware and the changing business requirements. So far, for the fault-prone modules various techniques have been proposed for predicting and forecasting in terms of performance evaluation. However, the kind of quality improvement and cost reduction as their actual need to meet the business objectives is rarely assessed.

Software failures are mainly caused by design deficiencies that occur when a software engineer, either misunderstood a specification or simply makes an error. It is estimated that 60-90% of current computer errors are caused due to the software failures [10],[12], [19]. These failure predictions has been studied in the context of fault-prone modules, self healing systems, developer information, maintenance models, etc., but a lot of things like modelling and weighting of the impact of different types of faults in different types of software systems must be explored for the fault severity in software development.

Performance requirements and reliability are fundamental to the development of high assurance systems. Based on the failure analysis it has proved a useful tool for detecting and preventing failures requirements early in the software lifecycle. Adapting a generic taxonomy fault, one is able to better prevent past mistakes and develop requirements specifications with less general failures. Fewer failures in the software specification, with respect to the requirements for performance and reliability, will result in high security and quality systems. The scope of this paper is to provide an overview of the mechanism in fault detection and techniques for the prevention of faults that can be followed in the quality software development process.

The following paper organizes in the seven sections. Section-2 and 3 discuss about software fault detection and software fault preventions mechanism. Section-4 presents fault prevention benefits and its limitation, section-5 presents related works and section-6 presents the conclusion.

II. Software Fault Detection Mechanism

A failure refers to any fault or imperfection in a work activity for a software product or software process cause due to an error, fault or failure. The IEEE Standards defines the terms *Error* as, a human action that leads to inaccurate results, Fault as, a wrong decision while understanding the information given to solve the problems or the application process. A single *error* can lead to one or more *faults* and a several *faults* can lead to failure. To avoid this failure in software products, faults detections activities are carried out in every phase of the software development life cycle based on their need and criticality.

A Monden et al. [1] proposes simulation model using fault prediction results for software testing to measure the cost effectiveness of test effort allocation strategies. The proposed model evaluates the number of qualified

faults in relates to resource allocation strategy, a set of modules, and the result of fault prediction. In a case study applying a small fault prediction system acceptance testing in the telecommunications industry, the results of our simulation model showed that the best strategy was to let the test effort is proportional to "number of failures expected in a module ". By using this strategy with our best prediction model of failure, the test effort reduced by 25%, while detecting as flawed normally found in testing, even if the company requires approximately 6% of the test effort for the collection of statistics, data cleansing and modelling.

A. Detection Using Automated Static Analysis

Automated Static Analysis (ASA) detection is mostly performed for the Manual Code analysis, which is one of the oldest practices are still practiced, but automated tools are increasingly used especially for the standard problems related to non-compliance faults possible memory leaks, variable usage etc. They have an essential place in the development phase because they save effort and significant resumption fault leakage test cycles. Findbugs, CheckStyle and PMD are some of the commonly used tools in the Java technology and there are many of these tools in all technologies. Although this plays an important role in the development cycle is not widely practiced in the maintenance mode. However, for systems that have compatible source for automatic static analysis detection tools can be used as a hygiene factor and good detection mechanism as any error introduced in the field is highly expensive. Maintenance cycle of ASA detection tools cannot find many flaws that may result in failures. A study on the effectiveness of ASA detection tools in the open source code reveals that less than 3% of the failures [2].

S Liu et al.[3] address static analysis technique problem that is commonly used for fault detection, but which suffers from a lack of rigor. It supports a systematic and rigorous inspection method takes advantage of the formal specification and analysis. The purpose of the method defined in the specification of a set of paths from each functional landscape program and the path specification of the program in every program contributes to the implementation of a functional landscape that is implemented correctly determine whether the inspection is used. Specification of functional scenarios to get the program paths, the paths linking scenarios, analyzing the paths against the scenarios, and the production of an inspection report, and a list of a systematic and automatic generation for inspection.

B. Detection Using Graph mining

Graph Mining is a dynamic control flow based approach that helps identify flaws that may be not crashing in nature. Use graphics calls are reduced by the simplicity in processing. The graph node represents the functions and a function call to another is represented by the edges. Edge weights are entered based on the calling frequencies. The variation in the frequency of call and change in the structure of call are potential failures. If there are problems in the data that is transmitted between the methods could also affect the graph of the named because of its implications.

C. Detection Using Classifiers

Classifiers based on the clustering algorithm and decision tree or neural network can be used to identify abnormal events of normal events for the detections. Classifiers are also formed by labelling defective tracks when a fault is observed. Some classifiers are commonly used NaiveBayes and bagging. Bayesian classification is a supervised learning method and a statistical method for classification. Representing an underlying probabilistic model that allows us to capture the uncertainty in the model of a reasoned determining the probabilities of outcomes. Recent research works [4] done in this area, without secondary supervision model that captures the normal code of behaviour probability distribution of each region is proposed to identify events when it behaves abnormally. This information is used to filter the labelling abnormality submitted to the ranking algorithm to focus on anomalous observations.

Machine learning classifiers [35] have recently introduced in the faults to predict changes in the source files. The classifier is first trained on software development, and then used to predict whether an upcoming change causes an error. Disadvantages of existing classifier-based bug prediction techniques are not enough power for practical use and slow prediction times due to a large number of machines learned functions.

S Shivaji et al. [5] investigates several feature selection techniques, which are generally for classification based fault prediction method using Naive Bayes and Support Vector Machine (SVM) classifiers. The techniques discard less important functions until optimal classification performance achieved. The total number of functions used for the formation is substantially reduced, often to less than 10 percent of the original. Both Naive Bayes and SVM with feature selection provides significant improvement in *Buggy F-measure* compared to the prior classification change failure prediction results compare to proposed in [6], work.

D. Detection Using Pattern Mining

Pattern based detection also the classifier based but uses unique iterative patterns for classification sequential data using the software trace analysis for failure detection. A set of discriminatory features capture repetitive series of events from the program execution traces first executed. Subsequently, the choice is made to select the best features for classification. Classifier model is trained with these sets of features that will be used to identify the failures. Processing pattern modelling allows together the analysis and improvement of processes, the work coordinate multiple people and tools to perform a task. Process modelling focuses generally on the normative process that is how transpires cooperation, if all goes as desired. Unfortunately, real-world processes rarely go that smoothly. A more complete analysis of the process requires that the process model and details of what to do when emergency situations occur.

B.S. Lerner et al. [7] have shown that in many cases there are abstract pattern to detect the relationship between the exception handling functions and the normative process. Just as object-oriented design patterns facilitate the development, documentation and maintenance of object-oriented programs, they believe that process patterns can facilitate the development, documentation and maintenance of process models. They focus on the exception handling pattern that we have observed over many years of process modelling. They also describe these patterns using three process modelling notations: UML 2.0 Activity Diagram [8], BPMN and Little-JIL [9]. They provide both the abstract structure of the pattern, as well as an example of the pattern is used. They present some preliminary statistical data to support the contention that these patterns are commonly found in practice, and represent in relation to their ability to use these patterns to discuss the relative merits of the three notations.

III. Software Fault Prevention Mechanism

In software development, many faults emerged during the development process. It is a mistake to believe that faults are injected into the beginning of the cycle and removed through the rest of the development process [10]. The faults occur all the way through the development process. Therefore, fault prevention becomes an essential part of improving the quality of software processes.

Fault prevention is a process of quality improvement which aims to identify common causes of faults and change the relevant processes to prevent the type of fault recurrence. It also increases the quality of a software product and reduces overall costs, time and resources. This ensures that a project can keep the time, cost and quality in balance. The purpose of fault prevention is to identify faults in the beginning of the life cycle and prevent it happening again so that the fault cannot appear again.

A. Importance of Fault Prevention

Faults prevention is an important activity in any software project development cycle. Most software project team focuses on fault detection and correction. Thus, fault prevention, often becomes a neglected component. Right from the early stages of the project to prevent faults from being introduced into the product that measure is therefore appropriate to make. Such measures are low cost, the total cost savings achieved due to profit later on stage are quite high compared to the cost of fixing faults. Thus, the time required for the analysis of faults in the early stages, reducing the cost and resources. Fault injection methods and processes enable fault prevention knowledge. After practicing this knowledge has improved quality. It also enhances overall productivity.

B. Activities in Fault Prevention

- **Fault Identification**

Fault can be a pre-planned activities aimed at highlighting the specific faults found. In general, faults can be identified in design review, code inspection, GUI Review, function and unit testing activities performed at different stages of software development life cycle. Once the faults are identified it will be classified using classification approach for the detection.

- **Fault Classification**

Classification of fault can be made using the general Orthogonal Defect Classification (ODC) technique [11] to find the fault group and it type. The ODC technique classifies the faults at the time when fault first occurs and when the fault gets fixed. The ODC methodology for each fault in orthogonal (mutually exclusive) to certain technology and some managerial Characteristics. These characteristics change through massive amounts of data can be analyzed and the root cause, the pattern to be able to access all the information on offer. Good action planning and tracking across with this fault reduction and can achieve high levels of learning.

Generally, important projects which are typically large projects needs to be classified in depth in order to get analyze and understand the faults, while the small and medium projects can be classified faults up to first level of ODC in order to save time and effort. The first level of ODC classifies the various types of faults in different stages of development requirement like Specification gathering, Logical Design, Testing and Documentation.

- **Fault Analysis**

Fault analysis is the continuous process for the quality improvement using fault data. Fault analysis generally classified in categories blame and direct process improvement efforts in order to attempt to identify possible causes. Root Cause Analysis (RCA) software fault has played a useful role in the analysis. RCA's goal to identify the root cause of faults and flaws that the source is eliminated so is to initiate action. To do this, faults one at a time are analyzed. Qualitative analysis is limited only by the limits of human investigative capacities. Qualitative analysis ultimately improves both the quality and productivity of software organization that provides feedback to the developers.

- **Fault Prevention**

Fault prevention is an important activity in any software project. Identify the cause of faults and fault prevention objective is to prevent them from recurring. Fault Prevention had suffered in the past to analyze the faults and faults in the future to prevent the occurrence of these types include special operations. Fault prevention software process to improve the quality of one or more phases of the software life cycle can be applied.

The benefits of analysis software faults and failures are widely recognized. However, a detailed study based on concrete data is rare. *M Hamill* et al. [12] analyze the fault and failure data from two large, real-world case studies. They specifically discuss the lead of software failure using localization of faults and different faults due to distribution. The results show that individual faults are caused often distributed through multiple errors in the entire system. This observation is important because it does not support multiple uses heuristics and assumptions about the past. Moreover, it is clear that the search for and fixing errors, such software errors that result in large, complex systems are often in spite of the advances in software development difficult and challenging tasks.

IV. Faults Prevention Benefits And Limitations

Fault prevention strategies exist, but reflect a high level of test maturity discipline associated with the testing effort represents the most cost-effective expenditure. To detect errors in the development life cycle from design to implement code specifications require that helps to prevent the escape of errors. Therefore, test strategies can be classified into two different categories as, fault detection technologies and fault prevention technologies.

Fault prevention efforts over a period of application development provide major cost and time savings. Thus it is also important, reduces the number of faults for reconstruction brings cost reduction, it is easy to maintain port and reuse makes. It is also necessary for the organization to develop high-quality systems in less time and provides resources, makes the system reliable. Faults which in turn increases productivity preventive measures are identified, based on which they have been injected to the life cycle stage can be traced back. A corrective measure for the promotion of knowledge of lessons learned between projects is a mechanism.

The lack of specific domain knowledge, where new and diverse domain software is a need to develop and implement. In many occasions, appropriate quality requirements specified are not in the first place. The inspection operation is labour intensive and requires high skill. Sometimes well-developed quality measurement may not have been identified at design time.

V. Related Works

No single software fault detection technique is capable of addressing all concerns in error detection. Similar software reviews and testing, static analysis tools (or automated static analysis) can be used to remove faults before a software product release. Inspection, prototyping, testing and proofs of correctness are several approaches to identify faults. Formal inspections to identify faults in the early stages of developing the most effective and expensive quality assurance techniques. Prototype through several requirements clearly helps to overcome the faults which are understood. Testing is one of the least effective techniques. May escape detection in the early stages, which is to blame, those tests could be detected in time. The accuracy proofs especially on the coding level are a good means of detection. Accuracy in manufacturing the most effective and economical way of building software.

J Zhang et al. [13] determine the extent to which automated static analysis can in the economic production to help a high quality product, they have static analysis and examine errors and customer reported losses for the three major in developed industrial software systems analyzed at Nortel Networks, The data show that automated static analysis is an affordable means of software error detection. Using orthogonal defect classification scheme, they found that automated static analysis effectively in identifying and mapping error checking, so that subsequent software production phases to focus on more complex, functional and algorithmic error. Much of the shortcomings that seem determined by automated static analysis are produced by a few major types of programming errors and some of these types have the potential to cause security vulnerabilities.

Statistical analysis results indicate the number of automated static analysis errors can be effective for identifying modules problem. The results analysis shows that the static analysis tools complement other error detection techniques for the economical production of high-quality software product.

Khoshgovar and Allen [14], [15] have proposed a model to check for software quality factors such as future fault density modules list. The inputs to the model are software complexity metrics such as LOC, number of unique operators and complexity. A stepwise regression is then performed to find weights for each factor. Briand et al.[16] using object-oriented metrics to predict classes that are likely to contain faults and used PCA in combination with logistic regression to predict failure-prone classes. *Morasca* and *Ruhe* [17] predicts risky faults modules using rough set theory and logistic regression in commercial software.

Over the years, several software techniques have been developed to support log-based fault analysis, the integration of state-of-the art gathering techniques to manipulate and to model the log data, for example MEADEP [18], Analyzing NOW [19], and SEC [20], [21]. However, log-based analysis is not supported by fully automated procedures so that most of the processing loads to analysts log is the often limited knowledge about the system. For instance, the authors in [22] have defined a complex algorithm for OS reboots from the log to identify on the basis of sequential analysis of log messages. Moreover, since an error activating multiple messages in the log cause a considerable effort to spent to the entries on the same mistake manifestation merged results [23], [24], [25]. Pre-processing tasks are critical to obtaining accurate failure analysis [26], [27].

While many case studies in the failure prediction in application for industry records reported [28], [29], [30] few studies have estimated achieved through early fault detection to reduce the test effort or increase the software quality. *Li* et al. [31] reported experience of application field fault prediction in ABB Inc. Their experiences are practical questions about how to select a suitable modelling method and how to evaluate the accuracy of the forecasts for several releases in the time period. They evaluated the usefulness of forecasts based on expert opinions. They reported that modules are identified vulnerable by experts as the failures of the top four fault prone identifies modules of the prediction model. They also reported that the module prioritization results were actually used by a test team to uncover the original be the low fault-prone additional faults in a module. Unfortunately it has no quantitative information on the effort for additional testing and the number of uncovered additional deficiencies required.

Mende and *Koschke* [32] and *Kamei* et. al [33] suggested that the efforts consciously measure to assess the failure prediction accuracy. While conventional valuation measures such as recall, precision, *Alberg* charts and ROC curves ignore the cost of quality assurance takes its action, the audit or review of a module is roughly expected to be proportional to the size. They took the advantage of their measure to the bottom to find the required prediction accuracy is required for the real testing.

C F. Kemerer et al. [34] studied the influence of the checking rate on software quality, while the controller for a comprehensive range of factors that can affect the analysis. The data comes from the Personal Software Process (PSP), which implements carried out inspections, the development group activities. In particular, the PSP design and code review rates correspond to the preparatory courses in inspections.

VI. Conclusion

Today there is an inherent need for software reliability is getting increased attention these days and highly fault tolerant system. In this survey paper, research on fault detection mechanism, as well as fault prevention mechanism in relation to the recent trend of the latest technologies have been discussed. There flaw detection and software systems used to diagnose the vast number of methods and techniques, but not every tech suits every system. Select technology system arrangement, size and complexity of adaptability and reliability targets, technology platform, driven by critical factors. Automated way to detect a tendency to higher levels in hybrid mining techniques and statistical models are in leaning toward more traditional systems-oriented solutions for diagnostics and prevention. Fault handling in modern day applications are in the early stages of research and the solution architecture try to build tolerance level as much as possible.

References

[1]. A Monden, T Hayashi, S Shinoda, K Shirai, J Yoshida, M Barker and K Matsumoto, "Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing", IEEE Transactions on Software Engineering, DOI-098-5589, 2013.
[2]. Fadi Wedyan, Dalal Alrmuny and James M. Bieman, "The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction", ICST '09. International Conference, vol., no., pp.141,150, 1-4 April 2009.
[3]. S Liu, Y Chen, F Nagoya and J A. McDermid, "Formal Specification-Based Inspection for Verification of Programs", IEEE Transactions on software engineering, vol. 38, no. 5, september/october 2012.
[4]. Bronevetsky, G.; Laguna, I.; de Supinski, B.R.; Bagchi, S., "Automatic fault characterization via abnormality-enhanced classification," Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on , vol., no., pp.1,12, 25-28 June 2012
[5]. 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, Vol. 39, No. 4, April-2013.
[6]. S. Kim, E. Whitehead Jr., and Y. Zhang, "Classifying Software Changes: Clean or Buggy?" IEEE Trans. Software Eng., vol. 34, no. 2, pp. 181-196, Mar./Apr. 2008.

[7]. B. S. Lerner, S Christov, L J. Osterweil, R Bendraou, U Kannengiesser and A Wise, "Exception Handling Patterns for Process Modeling", IEEE Transactions On Software Engineering, Vol. 36, No. 2, March/April 2010.
[8]. OMG, Unified Modelling Language, Superstructure Specification, Version 2.1.1, http://www.omg.org/spec/ UML/2.1.1/ Superstructure/PDF/, 2010.
[9]. A. Wise, "Little-JIL 1.5 Language Report", technical report, Dept. of Computer Science, Univ. of Massachusetts, 2006.
[10]. David Lo, Hong Cheng, Jiawei Han, SiauCheng Khoo and Chengnian Sun, "Classification of Software Behaviors for Failure Detection: A Discriminative Pattern Mining Approach", KDD '09 Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. Pages 557-566 ACM, USA, 2009.
[11]. Orthogonal Defect Classification – A concept for In-Process Measurements, IEEE Transactions on Software Engineering, SE-18.p.943-956.
[12]. M Hamill and K Goseva-Popstojanova, "Common Trends in Software Fault and Failure Data" IEEE Transactions on Software Engineering, Vol. 35, No. 4, July/August 2009.
[13]. J Zheng, L Williams, N Nagappan, W Snipes, J P. Hudepohl and M A. Vouk, "On the Value of Static Analysis for Fault Detection in Software", IEEE Transactions on Software Engineering, Vol. 32, No. 4, April 2006.
[14]. T. Khoshgoftaar and E. Allen, "Predicting the Order of FaultProne Modules in Legacy Software", Proc. Int'l Symp. Software Reliability Eng., pp. 344-353, 1998.
[15]. T. Khoshgoftaar and E. Allen, "Ordering Fault-Prone Software Modules", Software Quality J., vol. 11, no. 1, pp. 19-37, 2003.
[16]. L.C. Briand, J. Wiist, S.V. Ikonomovski, and H. Lounis, "Investigating Quality Factors in Object-Oriented Designs: An Industrial Case Study", Proc. Int'l Conf. Software Eng., pp. 345-354, 1999.
[17]. S. Morasca and G. Ruhe, "A Hybrid Approach to Analyze Empirical Software Engineering Data and Its Application to Predict Module Fault-Proneness in Maintenance", J. Systems Software, vol. 53, no. 3, pp. 225-237, 2000.
[18]. D. Tang, M. Hecht, J. Miller, and J. Handal, "Meadep: A Dependability Evaluation Tool for Engineers", IEEE Trans. Reliability, vol. 47, no. 4, pp. 443-450, Dec. 1998.
[19]. A. Thakur and R.K. Iyer, "Analyze-Now—An Environment for Collection and Analysis of Failures in a Networked of Workstations", IEEE Trans. Reliability, vol. 45, no. 4, pp. 561-570, Dec. 1996.
[20]. R. Vaarandi, "SEC—A Lightweight Event Correlation Tool", Proc. Workshop IP Operations and Management, 2002.
[21]. J.P. Rouillard, "Real-Time Log File Analysis Using the Simple Event Correlator (SEC)", Proc. USENIX Systems Administration Conf., 2004.
[22]. C. Simache and M. Kaaniche, "Availability Assessment of SunOS/ Solaris Unix Systems Based on Syslogd and Wtmpx Log Files: A Case Study", Proc. Pacific Rim Int'l Symp. Dependable Computing, pp. 49-56.
[23]. J.P. Hansen and D.P. Siewiorek, "Models for Time Coalescence in Event Logs", Proc. Int'l Symp. Fault-Tolerant Computing, pp. 221227, 1992.
[24]. Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R.K. Sahoo, "Bluegene/L Failure Analysis and Prediction Models", Proc. Int'l Conf. Dependable Systems and Networks, pp. 425-434, 2006.
[25]. A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R.K. Iyer, "Improving Log-Based Field Failure Data Analysis of Multi-Node Computing Systems", Proc. Int'l Conf. Dependable Systems and Networks, pp. 97-108, 2011.
[26]. D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving Software Diagnosability via Log Enhancement", Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems, pp. 3-14, 2011.
[27]. J.A. Duraes and H.S. Madeira, "Emulation of Software Faults: A Field Data Study and a Practical Approach", IEEE Trans. Software Eng., vol. 32, no. 11, pp. 849-867, Nov. 2006.
[28]. N. Ohlsson, and H. Alberg, "Predicting fault-prone software modules in telephone switches", IEEE Trans. Software Engineering, vol. 22, no. 12, pp. 886-894, 1996.
[29]. T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems", IEEE Trans. on Software Engineering, vol. 31, no. 4, pp. 340-355, 2005.
[30]. A. Tosun, B. Turhan, and A. Bener, "Practical considerations in deploying AI for defect prediction: a case study within the Turkish telecommunication industry", Proc. 5th Int'l Conf. on Predictor Models in Software Engineering (PROMISE'09), pp. 1-9, 2009.
[31]. P. L. Li, J. Herbsleb, M. Shaw, and B. Robinson, "Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc.", Proc. 28th Int'l Conf. on Software Engineering, pp. 413-422, 2006.
[32]. T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models", Proc. Int'l Conference on Predictor Models in Software Engineering (PROMISE'09), pp. 1–10, 2009.
[33]. Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort aware models", Proc. 26th IEEE Int'l Conference on Software Maintenance (ICSM2010), pp. 1-10, 2010.
[34]. C F. Kemerer and Mark C. Paulk, "The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data", IEEE Transactions on Software Engineering, Vol. 35, No. 4, July/August 2009.
[35]. V. Challagulla, F. Bastani, I. Yen, and R. Paul, "Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques", Proc. IEEE 10th Int'l Workshop Object-Oriented Real-Time Dependable Systems, pp. 263-270, 2005.