Defects Prediction and Prevention Approaches for Quality Software Development

Mashooque Ahmed Memon[1]
Department of Computing
Faculty of Engineering, Science and Technology (FEST) Karachi, Pakistan

Mujeeb-Ur-Rhman Magsi Baloch[2]
Department of Mathematics & Computer Science
University of Sindh Jamshero Hydrabad, Pakistan

Muniba Memon[3],
Syed Hyder Abbas Musavi[4]
Department of Computing
Faculty of Engineering, Science and Technology (FEST) Karachi, Pakistan

***Abstract*—The demand for distributed and complex business applications in the enterprise requires error-free and high-quality application systems. Unfortunately, most of the developed software contains certain defects which cause failure of a system. Such failures are unacceptable for the development in the critical or sensitive applications. This makes the development of high quality and defect free software extremely important in software development. It is important to better understand and compute the association among software defects and its failures for the effective prediction and elimination of these defects to decline the failure and improve software quality. This paper presents a review of software defects prediction and its prevention approaches for the quality software development. It also focuses a review on the potential and constraints of those mechanisms in quality product development and maintenance.**

***Keywords*—Software; defects; predictions; preventions; software development**

I. Introduction

The software is a single entity that has a strong impact on all characteristics of software development for different domains that includes defense, medicine, science, transport, telecommunications and others. The activities of all these domain sectors constantly require high-quality software for their exact needs for the performance [1]. Software quality means being an error-free product that produces predictable results and can be delivered within a time and cost constraints [2, 3]. As a result, it very important to have appropriate approaches to develop high-quality software that can meet the increasing needs in today's business world's. The past studies suggest that no single *defect detection technology* can solve all types of defects detection problems. So, this review focuses on the effectiveness and efficiency of the defect detection process to meet the quality enhancement and cost reduction.

A “defect” is some fault or imperfection in the operation of a software product or process as a result of an error, fault, or failure. The paradigm defines the term "error" as a human action that leads to inappropriate results, and a "defect" as an erroneous decision that results in inaccurate results for a solution to a problem. A single error can result in one or more failures, and multiple failures can cause a failure. To avoid such failures in software products, defect detection activities are performed at each stage of the SDLC, depending on the needs and criticalities of the development.

Software defect identifications models [2] are very weak because they have not been able to overcome the unknown relationship between the defects and failures. The relationships understanding among them are very difficult due to the diversity of defects and failures. The "Simplified assumptions" and "heuristics” methods are frequently utilized because of the associated failures associated with failures that lead to difficult tasks for the prediction. Therefore, having an accurate defect prediction model or process in software development can able to reduce high failures and advance the eminence of the software development [4, 5]. The main cause of software failures due to its design flaws which mostly caused by the software engineers due to the misunderstanding of the requirement specifications or developing a defective code. A review study on the various domain system failure estimation suggests that 90% of the failure is due to system defects [6, 7, 21].

The approaches of defect prevention are the process for improving software quality, the core objective of that is to identify frequent causes of defects and to amend the process to avoid this kind of the defect from importunate [8]. The purpose of preventing defects is to identify them at the commencement of the life cycle and avoid them from happening again so that the defects no longer occur. Based on defect analysis, it has established to be a constructive mechanism for detecting and preventing defect requirements at the beginning phase of the software lifecycle. By analyzing the general classified defects taxonomy and past errors it can be better prevented and reliable high performing systems can be developed [11]. In terms of performance and reliability requirements, a smaller number of failures in the software requirement will affect in improved secure and quality software systems. The scope of this paper is to present an insightful exploration of the mechanisms of defect detection and defect prevention approaches that can be pursued for the quality system development processes.

The following paper presents the importance of defect prediction in Section 2 and it approaches in Section 3. In the Section 4 it presents defect prevention methodology, and Section 5 discuss its importance. Section 6 concludes and summarizes the paper.

II. Importance of Defect Predictions

In literature many empirical studies and tools [1, 5, 7, 8, 18] are designed to identify the defects for the quality software development. But these approaches can be executed at multiple points during development, not testing, which usually only happens after the executable software module is produced. A key indicates in considerate the prospective value of evaluation is that it is approximated that defects that escape from one phase of the SDLC to another, it could take an instruction for the extent to restore in the next phase. As a result, the development cost, quality, and time of the software will be significantly impacted because it is implemented at the early of the development cycle.

The software defects observed in IBM operating system depend on the field data is presented in [8], which is being classified into 408 types of defects using an "Orthogonal Defect Classification (ODC)" [16]. This classification approach is to quantify the defect, failure relation and the accuracy of prediction, 668 defects are injected over 12-open source projects. The major goal of this quantification is to show a complex relationship between software defects and failure disabilities through identifying the availability of the multiple task, such as events, conditions, etc., but the ODC approach does not allow for multiple events or conditions analysis so, user must fix it manually.

TABLE I. A Summarization of Merit and Demerits of Exisisting Defect Prediction Approaches

REF#	Approach	Merits	Demerits
[13]	This paper has proposed seven test effort allocation strategies utilizing the complexity measure for Fault Prediction.	1. A software test simulation model based on defect prediction results for evaluating the cost-effectiveness of a test work distribution strategy. 2. The simulation model estimates the number of discoverable defects in relation to a given test resource, allocation strategy and a group of test modules for defect prediction. 3. The strategy with the best defect prediction model, test effort might be reduced by 25%, but still detected many of the defects commonly found in the test, but the company needed about 6% testing effort to collect metrics, organize data, and modeling.	1. This strategy shows the best failure prediction model but requires a high amount of test effort. 2. The results show that only the suitable test strategy with adequately high defect prediction accuracy can reduce the test workload through defect prediction.
[14]	Analysis of the Exception handling through patterns process modeling	1. It shows that in many cases, there are some abstract patterns to detect the relationship between exception handling functions and the specification process. 2. Emphasis is placed on the exception handling patterns observed in process modelling over the years and described using three types' process modelling notations.	1. It has found that the exception handling pattern described here is useful for increasing the level of abstraction of the process model. It provides a way to access exception handling by providing a framework of questions.
[15]	Defect and Failure data analysis.	1. This solution analyses the defect and failure data of real-system case studies. 2. Exclusively discuss the causes of software failures using other defects due to localization and distribution of defects. 3. The results show that entity faults are often reasoned for many faults spread all over the system.	1. It reveals the nature of defects and failures, and defects-defects are very beneficial.
[19]	It proposed a Specification-Based Inspection approach for the programs verification.	1. Systematic and rigorous inspection methods are available to take advantage of formal specifications and analysis. 2. The purpose of this method is to utilize checks to establish if each functional solution described in the specification is correctly executed by a group of program paths to contributes certain functional aspects of the specification. 3. The results show that this method perhaps more valuable at detecting "function-related faults" than PBR but may be somewhat ineffective in detecting implementation-related faults.	1. It does not provide evaluation support for powerful features related to testing, such as reading computer instructions, managing scans, and subsequent scans for code modifications.
[20]	Utilize the machine learning classifiers based on multi-function selection techniques and implement a classification-based bug prediction method using "Naive Bayes" and "Support Vector Machine (SVM)" classifiers for bug forecasting.	1. The research is generally applicable to a diversity of "feature selection techniques" based on classification-based error prediction methods. 2. Several feature selection techniques are studied, which are commonly used for classification-based defect prediction. 3. These techniques reject fewer essential features before achieving most constructive classification. The complete features utilized for training is significantly decreased below 10% of the original functionality. 4. Performance analysis of different numbers of features shows that even 1% of the original features can achieve powerful performance.	1. These techniques discard less important functions for achieving optimal classification performance. 2. A basic limitation of historically based error predictions, as there possibly recent types of errors that are not so far included in the training data.

In past years, several software technologies have been developed for the integration of state-of-the-art collection technologies that manipulate and model log-based error analysis and log data; for example, "MEADEP" [35], "NOW" [36], and "SEC" [37, 38]. However, since the log-based investigation is not supported by fully automated procedures, the processing load on most analysis loads is inadequate knowledge of the system. For example, a complex algorithm is defined for rebooting the OS in the log to identify based on sequential analysis of log messages. In addition, an error that activates multiple messages in the log causes considerable effort to use the entries for the same results of the error manifestation. Preprocessing tasks are crucial for accurate error analysis [6, 22, 27, 36].

A. Monden et al. [13] proposed a simulation model for software testing by means of defect prediction outcomes to measure the cost-effectiveness of the test assignment strategy. The proposed model assessment and resource allocation strategy, various qualified defects associated with a set of modules and defect prediction results. In a case study of the small failure prediction system recognition analysis in the telecommunications domain, the outcomes of the simulation model shows that the effective scheme is to make the test workload proportional to many failures likely in the module. Through using this strategy of the failure prediction model, the test work is reduced by 25%, while detecting defects that are usually found in the testing.

The merits and demerits of most relevant defect prediction approaches have been summarized in Table I.

III. Analysis of Defect Prediction Approaches

In this section, we discuss the various approaches and methods for defect prediction. Most of the approaches utilize machine learning and classification methodology to perform the prediction.

A. Defect Prediction based on Patterns

The Pattern-based detection is also based on classifiers but using a unique iterative pattern for classifying sequential data [11], software trace analysis is used for defect detection. A group of distinctive features captures a repeating sequence of actions from the program implementation trajectory that is executed first. Subsequently, the best attributes for classification are selected. Using those feature sets to train the classifier model, which will be used to identify defects. The pattern processing models allow the investigation and enhancement of processes together besides that working to coordinate multiple defects and tools to execute tasks. This kind of modeling usually focuses on the specification process, that is, how every work should execute as needed. Unfortunately, the real-world processes are rarely going well according to the need. A more comprehensive analysis of this kind of process still requires detailed information on the process model and their actions that should be taken in the event in case of failure.

B. S. Lerner et al. [14] have revealed that in numerous cases for the software defect handling, there are some abstract patterns that can detect the relationship between defect handling functions and specification procedures. As in an "object-oriented design patterns" makes the possibility of the "development", "documentation", and "maintenance of object-oriented programs", it can be considered that process patterns can assist the enhancement and maintenance of the process models. It focuses on the defect handling patterns which have observed in process modeling for many years. They also illustrate these patterns by means of three process modeling symbols with the "UML 2.0 Activity Diagram" [17], "BPMN" and "Little-JIL" [18]. It presents an abstract construction of the pattern, in addition to examples of usage patterns. It also discusses some preliminary statistics to support the arguments that are common in these models and represent their ability to use these patterns to consider the comparative merits among the symbols.

B. Defect Prediction based on Graph Mining

The methodology of Graphics mining is based on dynamic control flow that helps identify defects that might not crash a system [34]. Its functions as a simple processing through graph nodes calls to reduce the processing overhead during execution. A graph node characterizes a function and a function call to another function which is represented by an edge. The influence of everyone edge of a node is computed based on their calling frequency. The high variation in the frequency call and changes in the node structure of the graph may be the cause of the failure. If there is a problem with the data being reassigned between the methods, it may also affect the named graph because of its functional impact.

C. Defect Prediction based on ASA

The process of “Automatic Static Analysis (ASA)" [22], [27] based prediction is primarily used for physical code analysis, which is one of the oldest traditions still practiced, but automation tools are increasingly utilized for fundamental difficulty associated with "non-observance failures", "probable memory leaks", "variable usage", etc. They occupy an essential position in the development phase because they save effort and critical re-defect leak test cycles. There are many such tools which are commonly being used as, "Findbugs", "CheckStyle", and "PMD" based on Java technology. Even though this participates as a significant function in the development cycle, it is not widely used for the defect prediction in the maintenance cycle. However, systems with compatible sources for automated static analysis can be utilized as clean aspects for excellent detection mechanisms, because the errors introduced in the executing field scan are very expensive. The maintenance cycle of the ASA prediction tool does not find many defects that may perhaps guide to the failure. Research analysis for the efficiency of ASA detection tools over the open source code represents show < 3% of failures.

S. Liu et al. [19] have presented the solution to the problems of the statistical analysis system, which are generally utilized for defect detection, and suffering due to the requirement of rigidity. It sustains a methodical and strict inspection method that takes advantage of "formal-specification analysis". The intention of the process is to describe the specification of a group of routes from every tasks base program and the route specification of the program, where the program contribute to the execution of an appropriately implemented functional environment to determine whether to use the inspection or not. A systematic, auto-generated list of functional scenarios to obtain program paths, where each path has connected to scenarios and an inspection report generated.

C. F. Kemerer et al. [21] have studied the effects of inspection rates on software quality and studied the controller for a wide-ranging of a group of features that could influence the analysis. This data comes from the" personal software process (PSP)", performs inspections and performs development group activities. Specifically, the speed of the PSP design and code review corresponds to the preparation of the test.

J. Zhang et al. [22] has presented an enhancement to the automated static analysis which can help provide high-quality products in economic production, and they perform static analysis and check for errors and customer reports on three major sectors of the development of industrial software systems for "Nortel Networks analysis". This data shows an "automated static analysis (ASA)"for an appropriate means of detecting software errors. The automated static analysis using "Orthogonal Defective Classification “schemes is effective in identifying and mapping error probes so that subsequent software creation steps can target on more difficult, functional, and algorithmic errors. Most of the flaws that appear to be determined by automated static analysis are generated by some major type of programming error, and some of these types are likely to cause security vulnerabilities. The "Statistical analysis (SA)" outcome indicates that many automated SA errors can be effective in identifying module problems. Results analysis Static analysis tools show that it complements other error detection technologies to produce economical, high-quality software products.

D. Defect Prediction using Classifiers

A classifier based on a "clustering algorithm" and a "decision tree" or "neural network “are being utilized to recognize anomalous events of detected common incidents for the prediction [11], [12]. If a defect is found, the classifier labels the defect path to systematize the classifier. Some classification criteria generally use "NaiveBayes" and "Bagging”. The Bayesian classification is a "supervised learning method" and is a "statistical method" for classification [12]. It represents a basic probability model that can capture uncertainty in a model of reason that determines the probability of a result. A recent study [7], [8], [10], [12] in this province is proposed without a secondary supervisory model to capture the regular code of behavioral probability distributions in each region to recognize incidents when they behave abnormally. This information is utilized to filter more than the labeling gives to the positioning algorithm to focus on abnormal observations.

The prediction classifiers utilizing machine learning techniques [40] are recently introduced for the defects prediction in source files. A classifier is primary trained in the defects of software development and then used to prediction if the defect vision changes it will also cause errors. A disadvantage of the existing classifier-based defect prediction technique is that it does not have enough control for actual utilization due to the various machine learning functions and the prediction time is slow.

T. Mende et al. [23] has suggested that assessing the efforts consciously can measure the accuracy of defect prediction. The traditional evaluation methods such as "recall", "precision", "Alberg chart" and "ROC curve" ignore quality assurance costs, but the action is expected to be approximately proportional to the audit or review of the module. They took advantage of the measurement to the bottom to find that the required measurement accuracy was needed for the actual test.

S. Shivaji et al. [24] has typically considers numerous attribute collection techniques for classification-based error prediction methods that use "Naive Bayes" and "Support Vector Machine (SVM)" classifiers. This technology discards less significant functions in anticipation of the most constructive classification result to be achieved. The complete functions utilized in construction is considerably decreased, often down to below 10% of the original. Both "Naive Bayes" and "SVM" through attribute selection [9] present significant improvements in comparison to the F-measure of the classification in the failure prediction and results compared to those proposed in [25].

Although many case studies on failure prediction in industry record applications [28], [29], [30] few studies have been estimated by early failure detection to reduce test effort or improve software quality. P. L. Li et al. [26] reported on ABB's experience in applying field failure prediction. Their experience is about how to decide the precise modeling method and how to evaluate the actual accuracy of predictions for several versions of the time-period. They assessed the usefulness of the forecast depends on the professional view. They identified the module as vulnerable by an expert because it identified the top four error-prone errors that identify modules in the predictive model. In addition, the module priority results have been reported by the test team to be used to reveal additional errors that are probable to reason a low error in the module. Unfortunately, there is no quantitative information on the effort to further test and the number of additional leaks needed.

IV. Defect Prevention

During software development, many defects occurred during the period of the development process. It is a defect considering that defect which is injected at the early stage of a cycle and eliminates in respite of the development process [16], [31]. Therefore, error prevention is an essential element in enhancing the excellence of software processes.

Defect prevention is a quality improvement process aimed at identifying ordinary reasons of defects and altering related processes to prevent the type of error recurrence. It also improves the eminence of software products and reduces overall costs, time and resources. This allows the project to maintain a good balance of "time", "cost" and "quality". The intention of defect prevention is to recognize defects at the inauguration of the SDLC and prevent them from reoccurring so that defects do not reappear.

A. Methodology for Defect Prevention

Defect prevention is an important activity of SDLC. Most software project teams focus on defect detection and correction. Therefore, error prevention is often an ignored component. It is, therefore, appropriate to take steps to prevent defects from being commenced into the product at an early stage in the project. These measures are inexpensive and the total cost savings achieved by benefiting from the stage later are significantly higher than the cost of defect remediation. This saves costs and resources in the initial phases of defect analysis. The "Error injection methods" and processes facilitate knowledge of error prevention. After practicing this knowledge, quality has improved. It also improves overall productivity. The methodology for the defect prevention includes three phases as follows:

1) Identification of the defects: The identification of the defects can be pre-structured and designed according to the activities of specific failure defects being observed. Typically, defects can be identified in design reviews, code reviews, GUI reviews, functional and unit testing activities performed at different stages of the SDLC. In case of a defect is identified, the designed classifier classifies the defect utilizing the defined defect knowledge base. In case of having a vast defect knowledge base, it is important to analyze the failure defects through a continuous learning process to have an effective classification approach.

A model to examine software quality factors, such as a list of future defect density modules are proposed by T.Khoshgovar and E. Allen [31, 32]. The input to the model is a measure of "software complexity", such as LOC, the number, and complexity of distinctive operators. Then perform a stepwise regression to find the weight of each factor. L. C. Briand et al. [33] utilized "object-oriented metrics" to predict defect classes that might contain errors and used "PCA with logistic regression" to predict defect classes that are prone to errors. S. Morasca et al. [39] utilized a "rough set theory" and "logistic regression" to predict the possibility of the modules failure in commercial software.

2) Analysis of the Defects: The analysis of the defects is a continuous process for improving learning quality using defect or error data. Defect analysis generally categorized based on the process dependencies and condition process activities for the improvisation of defect identification and its possible cause for the prevention. The "Root cause analysis (RCA)" approach is an effective software defect analysis mechanism which is very useful in understanding the problems of a failure. The goal of the RCA is to recognize the root reason of defects and initiate the action of defects removal from the sources by analyzing each individual defect precisely. The qualitative analysis is inadequate only by the limitations of human investigation capabilities. This ultimately improves the quality and productivity of software organizations that provide feedback to developers.

3) Classification of the Defects: Defect classification can be done using common "Orthogonal Defect Classification (ODC)" techniques [16] to find defect groups and types. The ODC technology classifies defects when they occur first and when the defects are fixed. The ODC methodologies for specific technologies and some management characteristics and for each defect orthogonal can mutually exclude. These attributes provide access to all the information that comes from the root cause, pattern, and data through a tremendous amount of data that can be analyzed. A high-quality action preparation and tracking can reduce failures and enable high levels of learning.

In case of critical and large projects, it must be deeply classified to analyze and understand defects, and in the small projects, it can be classified as defects up to the initial level of the ODC to preserve time and effort. It classifies various types of defects at diverse phases of development requirements, such as specification collection, logical design, testing, and documentation.

Defect prevention has been encountered in the past to analyze future defects and to prevent these types of occurrences including special operations. Defect prevention software processes can be applied to improve the quality of one or more phases of the SDLC. From the beginning phases of the project, to prevent defects from being presented into the product, measurements are appropriate. Even these measures are low cost, and the total cost savings achieved due to the profit at the end of the phase are quite high compared to the cost of a fixed failure. Therefore, analyzing the time needed for failures at an early stage reduces costs and resources. The defect injection method and the process can realize defect prevention knowledge. After the practice, this knowledge improves the quality. It also increases overall productivity.

B. Importance of Defect Prevention

Defect mitigation strategies exist but reflect the most cost-effective expenditures reflecting the high-level test maturity principles associated with testing efforts. To detect defect errors in the development lifecycle for implementing code specifications in your design, you should avoid errors. Therefore, test strategies can be categorized into two categories: defect detection technology and defect prevention technology.

Defect prevention during application development can save significant cost and time. It is therefore also important to decrease the number of rebuild failures resulting in cost reductions, ease of maintenance of ports and reuse. Organizations must also develop high-quality systems and provide resources to make systems reliable in less time. Determining defects increases productivity precautions and can be traced back to the fact that these defects have been injected into the lifecycle phase.

The benefits of analysis software failures and defects are well known. However, there is little-detailed research based on concrete data. M. Hamill et al. [15] analyzes the defect and failure data of a two big real-time system case studies. They specifically discuss the causes of software defects by localizing and distributing defects and using other errors. The results show that individual failures occur frequently through multiple failures in the overall system. This inspection is significant because it does not sustain multiple-use heuristics and hypothesis about the precedent. Moreover, finding and fixing errors such as software errors that result in large, complex systems is often done despite the difficult and difficult development of software development.

Due to the lack of specific domain knowledge, the new and different domain software should be developed and implemented. In many cases, the appropriate quality requirements are not initially specified. Inspection work is labor intensive and requires a high level of skill. Sometimes a well-developed quality measurement may not have been identified at design time.

No software defect detection technology can solve all the problems in error detection. Similar software reviews and tests, static analysis tools (or automated static analysis) can be used to eliminate defects before the software product is released. Inspection, prototyping, testing, and proof of correctness are several ways to identify defects. Formal inspections to identify failures in the initial phases of developing the most efficient and expensive quality assurance techniques. The adoption of several required prototypes clearly helps to overcome the perceived deficiencies. Testing is one of the least efficient techniques. It may be possible to evade detection at an early stage, which is the culprit and can be found in time. Especially the accuracy at the coding level proves to be a good detection method. Create the most accurate and economical way to build software.

V. Conclusion

Nowadays, intrinsic demands for software reliability are growing, and high defect tolerance systems are attracting attention. This paper has discussed several defect detection mechanisms and defect prevention mechanisms in relation to recent trends in the latest technologies. This paper presented review of the importance of defect prediction and their various approaches. Although there are several methods and technologies that are used to analyze for defect detection in a software system, but not all technologies are suitable for all systems. This paper has discussed defect prediction based on patterns, graph mining, ASA, and using the classifiers. Defect prevention methodology through defect identification, analysis and classification and its importance in reducing the system failure have also been discussed. This paper concludes that selection of defect prediction and prevention should be based on the system size and its complexity to provide a more adaptable and reliable solution for defect handling and provide high-quality software.

References

[1] D. Bowes, S. Counsell, T. Hall, J. Petric, T. Shippey, "Getting Defect Prediction Into Industrial Practice: the ELFF Tool", IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 44-47, 2017.

[2] Q. Song, Y. Guo, M. Shepperd, "A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction", IEEE Transactions on Software Engineering, Pp. 1 - 1, 2018.

[3] Z. Li, X.-Y.Jing, X. Zhu, "Progress on approaches to software defect prediction", IET Software, Vol. 12(3), Pp. 161 - 175, 2018.

[4] A. Rahman, L. Williams, "Characterizing Defective Configuration Scripts Used for Continuous Deployment", IEEE 11th International Conference on Software Testing, Verification and Validation (ICST) Pp. 34 - 45, 2018.

[5] S. Huda, K. Liu, M. A.razek, A. Ibrahim, S. Alyahya, H. Al-Dossari, S. Ahmad, "An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction", IEEE Access, Vol. 6, Pp. 24184 - 24195, 2018.

[6] L. Pascarella, F. Palomba, A. Bacchelli, "Re-evaluating method-level bug prediction", IEEE 25th International Conf. on Software Analysis Evolution and Reengineering (SANER), pp. 592-601, 2018.

[7] R. Malhotra, L. Bahl, S. Sehgal, P. Priya, "Empirical comparison of machine learning algorithms for bug prediction in open source software", International Conference on Big Data Analytics and Computational Intelligence (ICBDAC), Pp. 40 - 45, 2017.

[8] A. Dehghan, A. Neal, K. Blincoe, J. Linaker, D. Damian, "Predicting Likelihood of Requirement Implementation within the Planned Iteration: An Empirical Study at IBM", IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Pp. 124 - 134, 2017

[9] X. Chen, Y. Shen, Z. Cui, X. Ju, "Applying Feature Selection to Software Defect Prediction Using Multi-objective Optimization", IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Vol. 2, Pp. 54 - 59, 2017.

[10] M. Lanza, A. Mocci, L. Ponzanelli, "The Tragedy of Defect Prediction, Prince of Empirical Software Engineering Research" IEEE Software, Vol. 33(6), Pp. 102 - 105, 2016.

[11] J. H. C. Wu, Jacky Keung, "Decision support for global software development with pattern discovery", 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Pp. 182 - 185, 2016.

[12] J. Yang, H. Qian, "Defect Prediction on Unlabeled Datasets by Using Unsupervised Clustering", IEEE 18th International Conference on High-Performance Computing and Communications, Pp. 465 - 472, 2016.

[13] 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.

[14] A. 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.

[15] 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.

[16] P. Tiejun, Z. Leina, F. Chengbin, "Defect Tracing System Based on Orthogonal Defect Classification", International Conference on Computer Science and Software Engineering, Vol. 2, Pp. 574 - 577, 2008.

[17] OMG, "Unified Modelling Language", Superstructure Specification, Version 2.1.1, http://www.omg.org/spec/ UML/2.1.1/ Superstructure/PDF/, 2010.

[18] A. Wise, "Little-JIL 1.5 Language Report", technical report, Dept. of Computer Science, Univ. of Massachusetts, 2006.

[19] 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, 2012.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] F. Wedyan, D. Alrmuny, and J. M. Bieman, "The Effectiveness of Automated Static Analysis Tools for Fault Detection and Refactoring Prediction", ICST '09. International Conf., Vol. 1(4), pp.141-150, 2009.

[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] 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.

[32] T. Khoshgoftaar and E. Allen, "Ordering Fault-Prone Software Modules", Software Quality J., vol. 11, no. 1, pp. 19-37, 2003.

[33] 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.

[34] A. A. S. Haghighi, M. A. Dezfuli and S. M. Fakhrahmad, "Applying mining schemes to software fault prediction: A proposed approach aimed at test cost reduction", In Proceedings of the World Congress on Engineering, pp.415-419, 2012.

[35] 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.

[36] 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.

[37] R. Vaarandi, "SEC-A Lightweight Event Correlation Tool", Proc. Workshop IP Operations and Management, 2002.

[38] J. P. Rouillard, "Real-Time Log File Analysis Using the Simple Event Correlator (SEC)", Proc. USENIX Systems Administration Conf., 2004.

[39] 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.

[40] 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.