Machine Learning– Defect Prevention of In-Appendage and Under Development Process of Quality Projects

B. Dhanalaxmi
Institute of Aeronautical Engineering College, India

Abstract– **Society has become increasingly dependent on software controlled systems (e.g., banking systems, nuclear power station control systems, and air traffic control systems). These systems have been growing in complexity – the number of lines of source code in the Space Shuttle, for instance, is estimated to be 10 million, and the number of lines of source code that will fly aboard Space Station Alpha has been estimated to be up to 100 million. As we become more dependent on software systems, and as they grow more complex, it becomes necessary to develop new methods to ensure that the systems perform reliably. One important aspect of ensuring reliability is being able to measure and predict the system's reliability accurately. The techniques currently being applied in the software industry are largely confined to the application of software reliability models during test. These are statistical models that take as their input failure history data (i.e., time since last failure, or number of failures discovered in an interval), and produce estimates of system reliability and failure intensity. To better control a system's quality, we need the ability to measure the system's reliability prior to test, when it is possible to influence the development process and change the system's structure. We develop a model for predicting the rate at which defects are inserted into a system, using measured changes in a system's structure and development process as predictors, and show how to: (i) Estimate the number of residual defects in any module at any time and, (ii) Determine whether additional resources should be allocated to finding and repairing defects in a module. In order to calibrate the model and estimate the number of remaining defects in a system, it is necessary to accurately identify and count the number of defects that have been introduced into a system. We develop a set of rules that can be used to count the number of defects that are present in the system, based on observed changes that have been made to the system as a result of repair actions. This paper identifies baseline procedures for verifying software for individual, small team, and large team development efforts for mission-critical and non-mission-critical software. It is based on defect-based inspections and basis path testing. Basis path testing provides a unified approach for performing unit, integration, and functional tests, whereas defect-based inspections are primarily used for verifying requirement and design documents. However, in situations where practitioners cannot afford to be as thorough as basis path testing permits, several heuristics are defined for prioritizing the remaining verification efforts and deciding which technique to apply. In addition, several studies are discussed that identify the relative merit of various verification techniques.**

Keywords– Software Quality, In-Appendage, Machine Learning and Defect-Tracking

I. INTRODUCTION

Quality of software is hectic in development, the rate of input produced per unit of quality output is considered as size. Software failures are costly to regarding the problems published regularly ranging from minor issues to the year 2000 thus a better understanding of software defects cause , the improvements in the areas of essential. Reliability of the software is an external attribute, the program has certain reliability from the user perspective and [6] internal attributes related to reliability are defects or faults.

Quality of the software is required, and not maintained how it effects the over all costs, we have so many tools and automated techniques have to take development make testing process with more efficient. A wide variety of proposed solutions the fundamental challenge of testing revealing other defects in freshly developed process or after major modifications.

While test automation is becoming increasingly popular approaches like Test-Driven Development and eXtreme Programming, empirical research shows that companies typically perform very little automated testing and most new defects are found by manual testing. The role of automation is emphasized in regression testing and it is best viewed as a way of removing the enactment of simple and repetitive tasks from human testers in order to free up time for creative manual testing. Interestingly, manual testing and especially test execution practices have been fairly little studied in the software engineering community. Testing research has focused on techniques for test case design, selection and

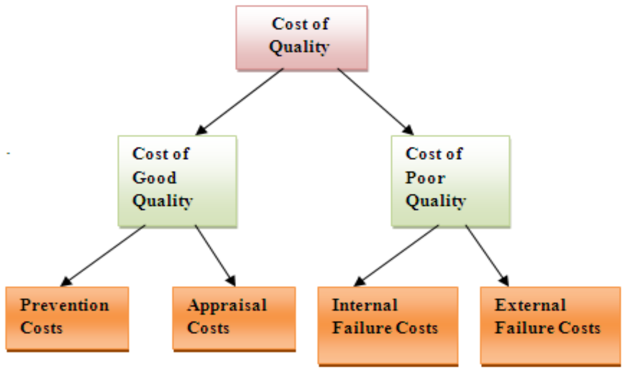


Fig. 1: Represents the Quality of Software cost of Attributes

prioritization, as well as on optimizing automated testing. However, we do not know, i.e., what factors affect the efficiency of manual testing, and how, or what practices industrial testers find useful. Previous research shows that aspects such as testers' skills and the type of the software have as strong an effect on test execution results as the test case design techniques. Software organizations today rarely meeting the high quality of attributes, to compensate for this lack of quality past research proposed four common [1] approaches to improving it.

One approach is to hire the best and effective personnel, although seldom is the criteria for selecting such people ever defined. Another way is to reuse software instead of developing it a novel. Unfortunately, few organizations have been able to develop general, reliable software that can be reused without significant modification. Yet another scheme is to develop software at higher levels of abstractions. However, it is still rare to convince others of the wisdom of this approach in light of anticipated system performance penalties. Therefore, the final approach is the one most commonly followed. It advocates the adoption of improved software processes that reduce the number of defects and the variability of them over time.

II. SURVEY OF DEFECT TECHNIQUES

To avoid each and every problem in development implanting such a thing means the system is too complex due to that here is the process defect prevention were focused on defect prediction and decide upon the team size of the testing resources required in order to complete the project on time and lot of effort were utilized in the debugging and get the defects eliminated with the advent of SDLC (System Development Life Cycle) processes many companies formulated their own defect prevention mechanisms and many studies were conducted towards defect prediction and prevention. The main modes of the defects would be Defect Origin, Types and Modes and how analysis of defects found in first iteration can provide feedback for defect prevention in later iterations, leading to quality and productivity improvement [2]. All the above methodologies lacked some dimension in the defect prevention process and needed more attention. In this study, we propose to combine the above methodologies used such as Iteration defect reduction, capturing defects at early stage and finding out defect prevention for better classified type of defects and have attempted to come out with a defect prevention cycle for continuous improvement of the Quality Processes and Defect Prevention.

A. Quality Prevention for Development Software

Important activity in any software project is bug prevention process. In most software organizations the project team focuses on defect detection and rework. Thus, defect prevention, often becomes a neglected component. It is therefore advisable to make measures that prevent the defect from being introduced in the product right from early stages of the project. While the cost of such measures are the minimal, the benefits derived due to overall cost saving are significantly higher compared to cost of fixing the defect at later stage. Thus analysis of the defects at early stages reduces the time, cost and the resources required. The knowledge of defect injecting methods and processes enable the defect prevention. Once this knowledge is practiced the quality is improved also enhances the total productivity.

Our work describes the approaches in predicting the number of defects to be discovered for a software product, particularly for software testing phase. It presents the overview of various techniques and models in predicting software defects across Software Development Life Cycle (SDLC). It then focuses on strategies in estimating defects for software testing phase using various models. Next, it describes the application and use of defect estimation with regard to software process improvement and software quality. Several critiques on defect prediction model are also presented. Finally, this paper illustrates the proposed model in predicting and estimating defects for software testing phase. And the approaches of defect prediction throughout Software Development Life Cycle (SDLC) are also explained. It consists of perspectives of defect and defect prediction, approaches and techniques of defect prediction as well as relationship of defect prediction with reliability.

As defect becomes the main focus of defect prediction, we should be able to distinguish between different defect severities, either major or minor defects. Minor should not be taken into considerations as it will inflate the estimation of product defects. From the observations done, most defect prediction depends on historical data. Furthermore, the techniques used to predict defect vary especially in term of data required (Clark and Zubrow, 2001). Prediction of defect can require little or more data. It also can rely on some work product characteristics or only use defect data. These differences in the quality of inputs used for predicting defects will determine the strengths and [3] weaknesses of a particular defect prediction. To start off with estimating defects, we must first aware on how defects are detected and generated. The purpose of understanding the defect detection is to identify the sources of defect or how defects are discovered. Defects can be detected either from verification and validation process or post-deployment.

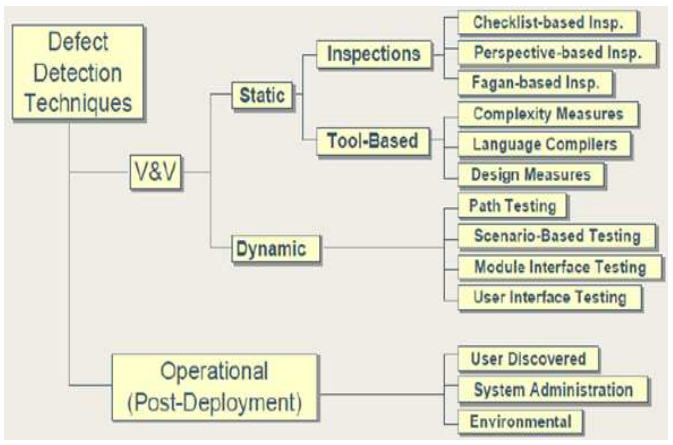


Fig. 2: Techniques of software Quality

Fig. 2 above summarizes the defect detection techniques as outlined in the studies by Clark and Zubrow. In general, defect prediction deals with estimating number of defects or faults. Defect prediction is usually used interchangeably with other terms such as defect estimation, fault prediction or fault estimation. Having the defect prediction helps in estimating the quality of software before being released and used by the users. The before analysis is required from three areas: predicting the number of defects in a system, estimating the reliability of systems in terms of time to failure, and understanding the impact of design and testing process on defect counts and defect densities. This defect prediction is expressed in a form of equations describing the defect inflow as a function of other selected measurements such as milestone completion status or lines of code (LOC), either from a short-term or long-term standpoint (Staron and Meding, 2007). Both standpoints help in monitoring the project status and project progress in developing software.

III. APPROACHES AND TECHNIQUES OF DEFECT PREDICTION

Various approaches and techniques have been formulated and applied in predicting number of defects throughout the entire SDLC. The techniques or approaches which are presented in a form of model or equation are developed according to several sources and metrics. Neil and Fenton (1999) presented their findings on how defects are predicted.

(i) First approach is prediction by using size and complexity metrics, in which it predicts defects directly based on program code, mostly towards lines of code and McCabe's Cyclomatic complexity. According to them, a study by Akiyama of Fujitsu, Japan showed that linear models of some simple metrics provide reasonable estimates for the total number of defects. From the four equations computed by him, one of them involves equation on lines of code (LOC) as below:

$$\text{Defect (D)} = 4.86 + 0.018 \text{ Lines of Code (L)}$$

They added on the argument by Gaffney that stated relationship between Defect (D) and Lines of Code (L) was not language dependent due to optimal size for individual modules with regard to defect. Lipow's data is used for the prediction:

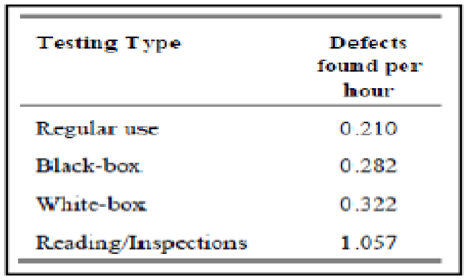
$$D = 4.2 + 0.0015\ L4/3$$

Further analysis was then conducted by Compton and Withrow who derived the polynomial equation, in which they concluded that the optimum size for an Ada module is 83 source statements with respect to minimizing error density. The equation is as below:

$$D = 0.069 + 0.00156\ L + 0.00000047\ L2$$

(ii) Second approach as outlined by Neil and Fenton is predicting defects using Function Point (FP). It is a measure of number of functionality in requirements for particular software. This [3] Albrecth Function Point describes defect density prediction by using metric extracted at specification stage due to believe function point-based metric is better than lines of code and is language independent.

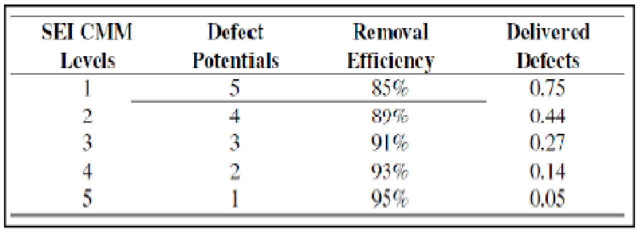
(iii) Testing metrics is another approach given by Neil and Fenton for predicting defects. This Testing metrics is another approach given by Neil and Fenton for predicting defects. This involves careful collection of data on defects found during inspection and testing phases. Test coverage measure is one of the testing [4] metrics used to predict defect via structural testing strategy. The resulting metric is called Test Effectiveness Ratio (TER) that covers either statement coverage, branch coverage or Linear Code Sequence and Jump coverage. Examples of how defects are found based on testing metrics is presented below:



Testing Type	Defects found per hour
Regular use	0.210
Black-box	0.282
White-box	0.322
Reading/Inspections	1.057

Fig. 3: A Defects based on testing metrics

Finally in their findings, Neil and Fenton described the usage of process quality data to predict the defects of software. This was expressed through the SEI Capability Maturity Model (CMM) ranking. The table below outlines the relationship between CMM levels and delivered defects.



SEI CMM Levels	Defect Potentials	Removal Efficiency	Delivered Defects
1	5	85%	0.75
2	4	89%	0.44
3	3	91%	0.27
4	2	93%	0.14
5	1	95%	0.05

Fig. 4: Relationship between CMM levels and delivered defects

For large software projects, studies by Staron and Meding have produced two types of prediction model which is defect inflow prediction (2007). One model is for short-term defect inflow prediction and another is for long-term defect inflow prediction. Historical data from the defect inflow trends and project plans is used to construct the short-term prediction model. From the data, multivariate linear regression prediction model is created, which then is applied in new projects to predict number of defects for a particular week. This multivariate regression model for short-term prediction is represented as an equation based on several independent variables as below:

$$Y = a_0x_0 + a_1x_1 + \ldots + a_nx_n$$

From the equation, values for 'a' is the coefficient calculated using statistical regression while method while 'x' is the independent variable. Based on the short-term prediction model, project team members can predict the number of inflow defects to be found in future week of the project execution.

Software Engineering Institute (SEI) of Carnegie Mellon University also conducted a study of the level of goodness for particular software. That study by Clark and Zubrow [4] emphasized on techniques in defect prediction. They categorized defect prediction techniques into three areas: project management, work product assessment and process improvement. Project management covers prediction techniques such as empirical defect prediction, defect discovery profile, COQUALMO and orthogonal defect classification. For work product assessment area, it involves fault proneness evaluation and capture/recapture analysis. As for process improvement area, defect prevention and statistical process control techniques are used.

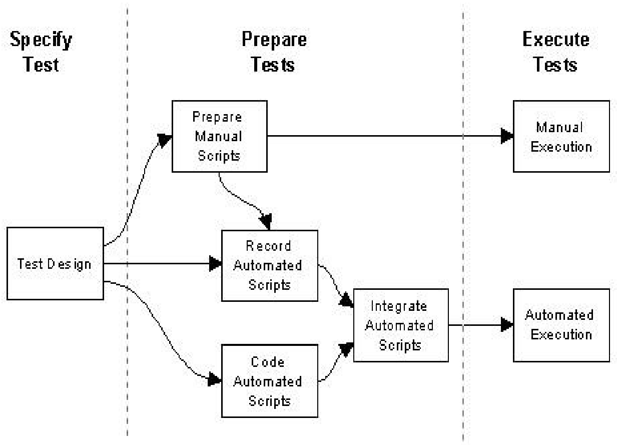


Fig. 5: Testing phase diagram

A. Defect Prediction in Testing Phase of Software Development Life Cycle

Approaches used in the previous findings of defect prediction mostly covered the potential number of defects to be found for all phases in Software Development Life Cycle but no specific prediction techniques explained for Testing phase. Although there are some techniques mentioned about defects to be found in System Test phase but the findings also take into account the defects to be found prior to and after System Test phase. The main intention is to understand more on the prediction techniques of defects to be found specifically for software testing phase of SDLC. Bertolino and Marchetti (2003) introduced a simple model called Bemar model. This model is used to predict the expected number of remaining failures in early test phases. It is quite simple since it predicts number of defects based on intervals of time between subsequent failures. The model is represented as below:

$$N_{F,k} = N_{FTI,k} \cdot E_k[F]$$

N_{FKIS} the number of failures for *k* test intervals, NFTI, k is the number of failures test intervals based on test information collected during *k* test intervals, and E_K [F] is the expectation of failures for *k* intervals. This Bemar model has been applied for functional testing and also operational test data. From the results, they concluded that the model assumes defects [5] detected are distributed over the whole test period. They also suggested that the model works well to complement reliability growth models. In the case study conducted at Sun Microsystems, Karcich, Cangussu and Earl proposed a state variable model called as the CDM Model (2003). CDM came from the developers' name of the state variable model for their Software Test Process: Cangussu, DeCarlo and Mathur. Besides using the model to control the test process using failure intensity as the control variable, the CDM model is also used to calculate the estimated number of total remaining defects.

B. Building Defect Prediction in Practice

It is very crucial to ensure the process of collecting data for predicting the defects is proper and accurate, so that the data that we used for analysis is correct. Generally, most of studies follow these steps or guidelines to statistically coming out with estimated defects:

1. Identify parameters or factors that have impact to defect injection in a software product.
2. Gather defect data for past projects in terms of total number of defects detected.
3. Analyze the correlation patterns between the parameters and the total defects found in past projects.
4. Estimate independent parameters for new project
5. Use Linear Regression to estimate total number of defects that may get injected based on the estimated independent parameters
6. Calculate the total number of latent defects.
7. Calculate efficiency required by project.
8. Calculate estimated defect rate for each period using Rayleigh Distribution.
9. Calculate estimated defect injection rate by phase based on project schedule.
10. Plot the S-shaped curve for defect detection pattern.
11. Compare Rayleigh curve and actual data to get quantitative estimate

The processes of developing prediction model based on Regression Analysis are as follows:

1. Gather data on given independent variables and correspondent dependent variables
2. Determine the form of equation to fit by plotting the dependent and independent data sets on a special graph such as scatter plot to shows the existence of statistical relationship
3. Fit an equation depending on number of independent variables either simple or multiple regression
4. Evaluate the fit using statistics such as Coefficient of Determination (R) or Standard Error of Estimate (SE)

C. Application of Defect Prediction

Defect prediction is used for various purposes throughout Software Development Life Cycle (SDLC). This is described in the Process Performance Model in which defect prediction model is one of the importance contributors. Process Performance Model predicts the effort, number or defect and other related data based on parameters such as schedule and size. One of the items in the quality planning is outlined by the experts with regard to process performance is to control number in User Acceptance Testing (UAT) phase. For this, it starts with predicting the total number of defects using defect prediction model, which then being adjusted according to project parameters such as customer quality goals, past data from similar project and type of development methodology used. Then, the defects are distributed amongst the phases in software life cycle.

Next, these distributed defects are adjusted for three things: to distribute defects early in the life cycle to achieve zero defects at acceptance phase, to distribute the remaining defects in other phases as per project scope and also to be used for verification and validation strategy which involves use of various type of test strategy to tackle more defects. From the result, project team should be able to derive several measures such as defect per function point per phase, defects per person month and also review effectiveness. The data will then be recorded and tracked. Defect prediction is also used to determine the reliability of software. This is because defect prediction is also part of the software reliability model. Software reliability model aims to estimate the reliability of the latent defects of software, especially when it is available to customers. The defects estimated across the SDLC provide a basis for describing the probability of the software operating in a given environment within the design range of input without failure.

Rayleigh Model is chosen to be the suitable software reliability model as it predicts the expected value of defect density at different stages of life cycle of the project. The equation presented in the Rayleigh Model is used to predict the number of defects over time. In order to determine the accuracy of the duration and magnitude of this Rayleigh Model, specific inputs must be selected. Having good inputs to the model allows accurate forecast for a specified scenario. Three main factors of the model are mentioned in several studies: source lines of code in a form of size required to build the software functionality, productivity index in a form of product efficiency and complexity as well as peak staffing in terms of human effort required to build and test the software.

The measurement of total defects likely to be occurred from the software being constructed is represented by the area bounded by the x-axis and the curve as depicted in the figure below:

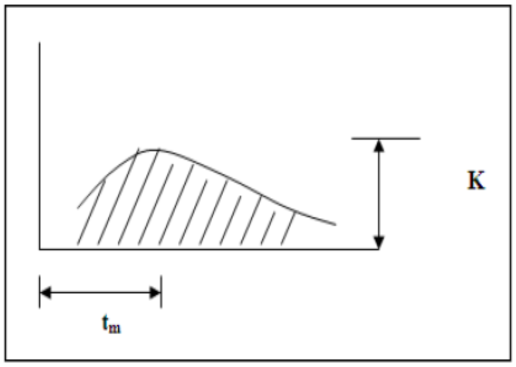
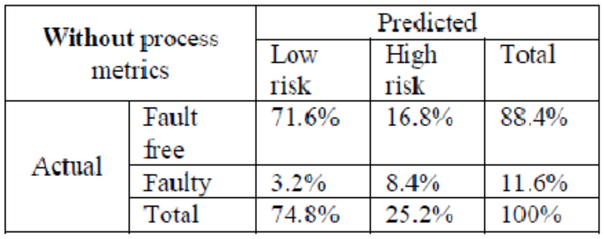


Fig. 6: Graphical representation of Rayleigh model parameters

From the above figure, an equation of Probability Density Function (PDF) is produced, which is F (t) = f (K, tm, t). *K* denotes cumulative defect density, *tm* represents actual time unit while *t* is the time at the peak of the curve. Good software maintenance also depends on good prediction model. Selecting good defect prediction model is important for pricing maintenance contracts and insurance (Li, Shaw and Herbsleb, 2003). It also helps in predicting support costs for software including maintenance staffing. Defect prediction model helps in planning the maintenance activities and timing for resolving reported defects. This is because a good model should be able to simulate occurrences of similar defects in the field. The essential thing to consider here is the different type of operational setting in which the model is applied to. The model should be able to work in environment of user-reported defects, widely-used systems, multi-release systems or commercial systems so that suitable maintenance activities can be adopted.

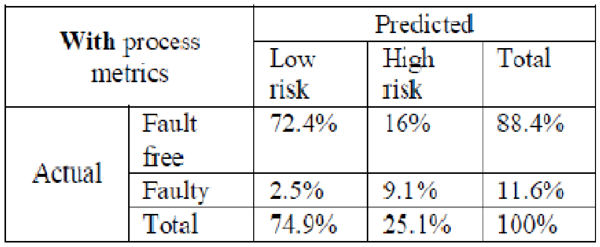
D. *Enhancement to Defect Prediction*

One approach to enhance the defect prediction is by using the process metrics. Process metrics or process data covers the data that is gathered in and by the problem tracking system and the configuration management system (Kaszycki, 1999). The data can be in a form of number of changes since last release, number of faults found since last release, number of different developers who turned over, versions of this module since last release or number of features that were added that affected this module. By using process metrics, it contributes to developing a higher accuracy of defect prediction model as well as helps in earlier detection of defect in the development process. Figures 7 and 8 below depict the differences between prediction without process metrics and prediction with metrics.



Without process metrics		Predicted		
		Low risk	High risk	Total
Actual	Fault free	71.6%	16.8%	88.4%
	Faulty	3.2%	8.4%	11.6%
	Total	74.8%	25.2%	100%

Fig. 7: Prediction without process metrics



With process metrics		Predicted		
		Low risk	High risk	Total
Actual	Fault free	72.4%	16%	88.4%
	Faulty	2.5%	9.1%	11.6%
	Total	74.9%	25.1%	100%

Fig. 8: Prediction with process metrics

Another approach to the enhanced defect prediction is through advanced model.

This is achieved via phase level Bayesian Networks (BN) for defect prediction. The objective is to predict defects and defect rates at different periods across software development project based on information available at any stage of development and testing (Neil, 2006). This advanced model takes into account several things: how big the software is, how good the development process is, how good the testing process is and also chances of successfully of removing defects.

E. MIMOS Software Production Process

MIMOS is a Software company which achieved the Level 4 CMMi(industry practice) compliance. As presented in Figure 9 below, testing team involves in all review session for each phase, starting from planning until end of system testing phase throughout the software production process. Test engineers involve in reviewing planning document, requirement analysis document, design document, test planning document and test cases. The software production process is governed by project management, quality management, configuration and change management, integral and support as well as process improvement initiatives, which CMMi. From Figure 9, the area of study is the functional or system test phase. In order to perform further analysis and establish defect prediction model for system test phase, faults and errors captured in previous phases prior to testing phase must be considered and investigated.

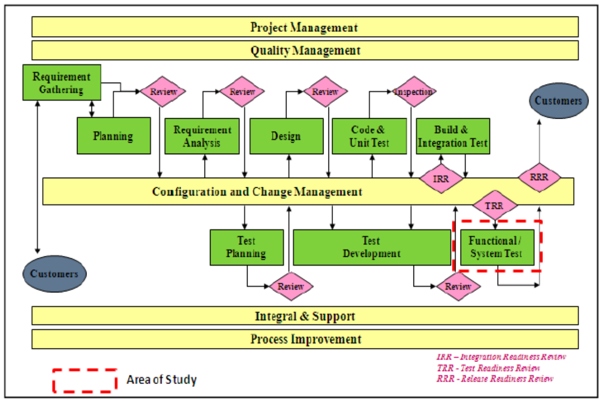


Fig. 9: MIMOS software production process

Technically, in building the defect prediction model, it is observed that many factors contribute to the defect discovery in testing phase. Obviously, faults in requirement, design and coding as well as in-process faults have their own relationship with defects. Code size in a form of kilo lines of code also affects the number of defects found in testing phase. By extracting the correct data from right sources, we will be able to conduct proper and detailed analysis on the identified factors while at the same time, proves that all factors must be considered in predicting defects for testing phase. The research shows that defect prediction model provides strong contribution to zero-known post release defects of particular software product since testing is the last gate in the process before the software can be said as fit for release and use. Test engineers will discover as many defects as possible to ensure all defects are contained within the testing phase and not escaping to the end-user. Additionally, having a predicted number of defects allows for better resource utilization of test engineers for a project by allocating appropriate number of testers to test the software.

Better test strategy and wider test coverage could be implemented by having predicted number of defects. This can be achieved practically since every test engineer will be aware of the potential defects that they will discover. The tolerance of 10% lesser or 10% greater of actual defects found against the estimated defects could be their guide in testing the software product. Indirectly, having estimated number of defects in testing phase promotes the initiatives of the whole software development process, especially in ensuring stability of development effort in releasing a software product.

IV. PROBLEM DEFINITION

In software Development the phase Testing is a executing program which discovers the error, defect, bugs. Preparing test-cases gathering requirements then apply the test manually or automated approach again the process becomes development, to avoid these process we propose a In-appendage which is invention of effective action based defect detection & Prevention Technique under development process.

In-Appendage is solution to find the defects in development phase which usually reduces the time and risk. In Development life cycle project suppose we have eight modules then other module is the appendage, ninth module appendage discovers the defects or bugs in before modules using machine learning techniques in Data mining.

A. Work Flow Stages

(i) Defect Identification: Defects are found by preplanned activities specifically intended to uncover defects. In general, defects are identified at various stages of software life cycle through activities like Design review, Code Inspection, GUI review, function and unit testing. Once defects are identified they are then classified using first level of Orthogonal Defect Classification.

(ii) Defect Classification: Orthogonal Defect Classification (ODC) is the most prevailing technique for identifying defects wherein defects are grouped into types rather than considered independently. ODC classifies defect at two different points in time Time when the defect was first detected – Opener Section Time when the defect got fixed – Closer Section ODC methodology classifies each defect into orthogonal (mutually exclusive) attributes some technical and some managerial. These attributes provide all the information to be able to shift through the enormous volume of data and arrive at patterns on which root-cause analysis can be done. This coupled with good action planning and tracking can achieve high degree of defect reduction and cross learning.

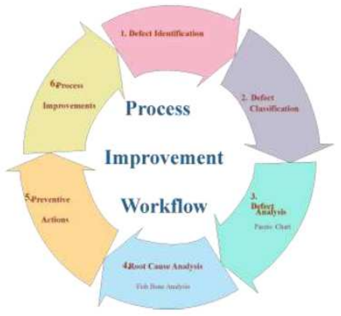


Fig. 10: Process Improvement Work flow

For small and medium projects, in order to save time and effort, the defects can be classified up to first level of ODC while critical projects typically large projects needs the defects to be classified deeply in order to get analyze and understand defects. In this paper, the project that is selected for analysis being a project coming under the category of small and medium size project, the analysis of defect is done by using first level of ODC defect classification. First level of ODC includes classifying the defects under various defect types like Requirements, Design, Logical (Logical defects are found by testing the [7] code using functional/unit testing), and Documentation. Defects are classified under these types and then analysis of defects is carried out.

B. Defect Analysis

Defect Analysis is using defects as data for continuous quality improvement. Defect analysis generally seeks to classify defects into categories and identify possible causes in order to direct process improvement efforts. Root Cause Analysis (RCA) has played useful roles in the analysis of software defects. The goal of RCA is to identify the root cause of defects and initiate actions so that the source of defects is eliminated. To do so, defects are analyzed, one at a time. The analysis is qualitative [8] and only limited by the range of human investigative capabilities. The qualitative analysis provides feedback to the developers that eventually improve both the quality and the productivity of the software organization.

Defect Prevention: Defect prevention is an important activity in any software project. The purpose of Defect Prevention is to identify the cause of defects and prevent them from recurring. Defect Prevention involves analyzing defects that were encountered in the past and taking specific actions to prevent the occurrence of those types of defects in the future. Defect Prevention can be applied to one or more phases of the software lifecycle to improve software process quality.

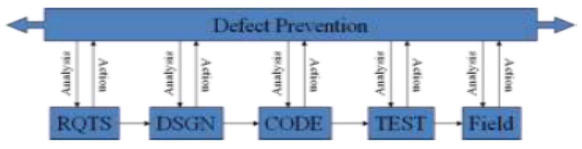


Fig. 11: Defect Prevention in Software Lifecycle

Process Improvement: The suggested preventive actions are implemented by rewriting the existing quality manuals and tweaking the SDLC processes and come out with a improved SDLC processes and documents. Next set of projects follow the revised quality processes there by effectively all the preventive actions are followed meticulously.

C. Project Defect Data

Information like number of lines of code (KLOC) produced by the software, number of defects and the number of man hours spent in the project are collected in order to know the defect data in the project. Defect density is a measure of the total number of defects in a project divided by the size of the software being measured.

Defect Density (DD) = Number of defects / size (kloc) –

Defect density is calculated to track the impact of defect reduction and to judge the quality improvement on the project that has implemented defect preventive action with the project that did

Defect Pareto Chart: After defects are logged and documented, the next step is to review and analyze them using root cause analysis techniques. Before root cause analysis is being carried out, A Pareto chart is prepared to show the defect type with the highest frequency of occurrence of defects – the target.

Root Cause Analysis: Root-cause analysis is the process of finding the activity or process which causes the defects and find out ways of eliminating or reducing the effect of that by providing remedial measures. The root cause analysis of a defect is driven by two key principles: Reducing the defects to improve the quality: The analysis should lead to implementing changes in processes that help prevent defects in the formation stage itself and ensure their early detection in case it is re-occurring.

Utilizing local and third party expertise: The people who really understand what went wrong should be present to analyze processes prevalent in that organization along with third party experts. A healthy debate ensures all possibilities are reviewed, analyzed and the best possible actions are arrived by consensus [5]. With these guidelines, defects are analyzed to determine their origins. A collection of such causes will help in doing the root cause analysis. One of the tools used to facilitate root cause analysis is a simple graphical technique called cause-and-effect diagram/ fishbone diagram which is drawn for sorting and relating factors that contribute to a given situation.

Preventive Action: A standard brainstorming procedure was followed to do root cause analysis. First all the possible causes were identified from the cause-and-effect diagram and debated among the team and all suggestions were listed, then the ones that were identified as the main reasons for causes were separated out. For these causes, possible preventive actions were discussed and finally agreed among project team members

D. A Model for the Rate of Defect Insertion

We propose to model the rate at which defects are inserted into a software system. In general, we will model the rate as functions of the measured structural change in a software system over any given development increment and the measured changes in the development process over that same increment, given by *ix* = *f x* (D*sx*, D*d x*), where *ix* is the rates at which defects are inserted and deleted when **x** defects are already in the system, and *f x* (*sx d x*) D ,D is a function of the measured structural change, D*sx* , and [8] the measured development process change, D*d x* , over a development increment at the start of which **x** defects were in the system. The function *f x* (*sx d x*) D ,D is not required to be constrained to any particular form, and may indeed vary from development phase to development phase within a software development effort.

However, previous work by experts shows that during the implementation phase, the correlation of measurements of system structure (but not structural change) and the system's defect content is 0.90, and that the relationship between measurements of system structure and the defect content is linear. For the implementation phase, then, we will take as our starting point the hypothesis that the rate of fault insertion is linearly related to the measured structural change and development process change during a development increment: *ix* = *k x*D*sx* + *k x* D*d x* 0, 1, where D*sx* and D*d x* are as defined above, and *k*0,*x* and *k*1,*x* are constants relating the measured structural and development process change to the rates of defect insertion and removal at the start of a development increment in which the system contains **x** defects. In the simplest case, the constants *k*0,*x* and *k*1,*x* would be the same for all values of **x**.

Furthermore, if the development process were to remain constant across a particular development phase, the term for the effects of change to the development process, *k*1,*x* D*d x* , would assume a value of 0. The effects of the [9] development process would be taken into account in the constants *k*0,*x* . This would make it particularly simple to estimate the number of defects in the system at any given time. If, on the other hand, the rate at which defects were inserted into the system were to vary with the number of defects already in the system, estimating the number of defects in the system at any time would be more complicated.

V. CONCLUSION

From the above discussion, we draw several conclusions with regard to new approach of defect prediction model for software testing phase in SDLC. First, it is important to set the clear objective of what the proposed model need to achieve when it is implemented in real software development operations, which is to be able to estimate total number of defects to be discovered in software testing phase. Getting started with sample technique will do. Second, identification and collection of appropriate factors data that has strong significance with defect need is very essential in defect prediction by following proper steps or processes, especially historical data. Whatever data that is available in place could help in determining the suitable prediction technique. This is because the historical data may drive the model selection. This bring to third conclusion in which the statistical relationship between the factors and defects must be established in coming out with the model to determine the correlation between those parameters. Instead of just focusing on fixing defects, analysis on the patterns against defects can be carried out.

Fourth, verification of the model, in which in a form of equation, must be performed to ensure the model works and suitable with the internal software production process. Implementation of defect preventive action not only helps to give a quality project, but it is also a valuable investment. Defect prevention practices enhance the ability of software developers to learn from those errors and, more importantly, learn from the mistakes of others. The benefits of adopting defect prevention strategy would be enormous and to list a few, Defect prevention reduces development time and cost, increases customer satisfaction, reduces rework effort, thereby decreases cost and improves product quality. This paper confirms to implementation of first level of Orthogonal Defect Classification (ODC) for defect classification. To gain a deeper understanding about the defect, the defects are to be classified by implementing ODC to next level. Analysis of ODC classified data helps in getting better defect preventive ideas that would further improve the software quality process.

REFERENCES

[1] Clark, B. and Zubrow, D. (2001). How Good is the Software: A Review of Defect Prediction Techniques. Software Engineering Symposium. Carnegie Mellon University.

[2] Fenton, N.E. and Neil, M. (1999). A Critique of Software Defect Prediction Models. IEEE Transactions On Software Engineering. Volume 25, No.5.

[3] Grottke, M. and Dussa-Zieger, K (2001). Prediction of Software Failures Based on Systematic Testing. Ninth European Conference on Software Testing Analysis and Review. Stockholm.

[4] Mohanty, B. and Mohapatra, S. (2001). Defect Prevention Through Defect Prediction: A Case Study at Infosys. Proceedings of IEEE International Conference on Software Maintenance.

[5] Nayak, V. and Naidya, D. (2003). Defect Estimation Strategies. Patni ComputerSystems Liited. Mumbai.

[6] Neuendorf, S. (2004). Prediction of Software Defects. SASQAG 2004.

[7] Ostrand, J.T. and Weyuker, E.J. (2007). How to Measure Success of Fault Prediction Models. SOQUA '07. 25-30.

[8] Ostrand, T.J., Weyuker, E.J., Bell, R.M. and Ostrand, R.C. (2005). A Different View of Fault Prediction. Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC '05).

[9] Rana, Z.A., Shamail, S. and Awais, M.M. (2008). Towards a Generic Model for Software Quality Prediction. WoSQ '08. Leipziq.

[10] Thangarajan, M. and Biswas, B. (2002). Software Reliability Prediction Model. Tata Elxsi Whitepaper.

B. Dhanalaxmi M.Tech Software Engineering from Gurunank College of Engineering. Currently she is Asst Prof at Institute of Aeronautical Engineering College, has guided many UG & PG students. Her areas of research include Software Engineering , Quality Testing, Software Project Management Network Security.