ISSN 2277-3061

Software Defect Prevention through Orthogonal Defect Classification (ODC)

Sakthi Kumaresh, R Baskaran
Asso. Prof., Dept. of Computer Science,
MOP Vaishnav college for Women, Chennai
Asso. Prof., Dept. of Computer Science and Engineering,
Anna University, Chennai.

ABSTRACT

"Quality is never an accident; it is always the result of intelligent effort" [10]. In the process of making quality software product, it is necessary to have effective defect prevention process, which will minimize the risk of making defects /errors in software deliverables. An ideal approach would involve effective software development process with an integrated defect prevention process. This paper presents a Defect Prevention Model in which Defect Prevention Process(DPP) is integrated into software development life cycle to reduce the defects at early stages itself, thereby reducing the defect arrival rate as the project progresses to the subsequent stages. Orthogonal Defect Classification (ODC) scheme involving defect trigger, defect type etc. are discussed in this work to illustrate how ODC can be used in the defect prevention process. ODC can be used to measure development progress with respect to product quality and identify process problems, which will help to come out with "Best Practices" to be followed to eradicate the defects in the subsequent projects.

Indexing terms/Keywords

Defect, Defect Prevention Model, Defect Trigger, Orthogonal Defect Classification.

Council for Innovative Research
Peer Review Research Publishing System
Journal: International Journal of Computers & Technology
Vol 11, No.3
editor@cirworld.com
www.cirworld.com, member.cirworld.com

ISSN 2277-3061

INTRODUCTION

Software defect can be defined as “Undesirable events occurring in the software development process which in turn causes delay and lowers the quality of the software”. A defect in software may be due to some type of error or fault. Usually these faults are a result of human mistake, but sometimes they are caused by faulty development tools, vague customer requirements, incorrect design, and wrong test cases etc. The powers of man are not so extra-ordinary to never make mistakes; but from their errors and mistakes the wise and good learn wisdom for a better future [4]. It is important to implement a process that individuals and teams can make use of, to learn from their mistakes. A fundamental aspect of this learning is the classification of defects using orthogonal defect classification. With a structured classification scheme, an organization can analyze and learn about the types of defects that have been discovered and their relative frequencies. Such classification scheme provides insight into what improvements are needed to prevent or mitigate those defects in the future.

Defect Prevention is the process of improving quality and productivity by preventing the injection of defects into a product. This paper highlights the various components involved at every stage of software development, and the steps needed **to** implement the defect prevention process. The defect prevention model proposed in this study is a process to continually improve the development process. DPP is integrated into every stage of the development process. This approach ensures that meaningful discussion takes place when it is fresh in everyone’s mind. It focuses on defect related actions and process oriented preventive actions. This paper makes an attempt to adopt defect prevention process in mini-ERP project of small and medium scale enterprise and the results were obtained.

LITERATURE SURVEY

The earlier studies in 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 elimination instead of prevention. With the enhancements to SDLC processes many companies have formulated their own defect prevention solutions. One study by Natesan Karthkeyan [2] was to analyse various defect prevention techniques, its advantages and disadvantages, their cost analysis vis-a-vis alternate solutions. Research executed by Ms Prakriti Trivedi[3] uses a model for defect prevention using ODC as an approach for defect classification and prevention. Another paper by Mohd. Faizan[6] have also analysed various defect prevention techniques with restrictions to recent trends. The paper by Norm Bridge[5] has presented a framework developed by IBM for classifying and analyzing defect data collected during software development and describes how Orthogonal Defect Classification (ODC) can be used to measure development progress with respect to product quality and identify process problems. The paper by Prof. Pankaj Jalote[7] have focussed mostly on monitoring of quality control activities, like defect prevention, for ensuring high quality, are used. In another study by Prof Suma [1] the defect prevention issues faced by Small and Medium scale industries has been analysed and solutions have been suggested. In this paper, we propose to combine and enhance the above methodologies used, such as ODC for defect classification and analyse the defect patterns to arrive at early stage defect reduction. This paper attempts to bring best practices for defect prevention based on this mechanism for small and medium scale enterprise to implement it easily and effectively.

Defect Prevention Model With ODC

In a typical software development project, the test team becomes involved late in the process to find defects and “test quality into the software.” Unfortunately, the later a defect is discovered, the more expensive it is to repair and the greater the cost and impact on the overall project, just like the saying “A stitch in time saves nine”. Consequently, if defects cannot be avoided altogether, a fundamental goal of a successful defect prevention effort is to move quality verification and improvement to an earlier stage in the software development cycle. Focusing on quality in the planning, design, and early development stages pays big dividends later in the cycle. By moving quality assessment and improvement “upstream” in the software development process [4], the test team can focus more on the end user experience and on integration-level testing, rather than finding design or functional errors.

This paper describes a new type of model which integrates the Software Lifecycle development with defect prevention process. Figure 1 show the defect prevention model proposed in this paper. The idea behind this model is that Defect prevention process should be incorporated at each phase of software development life cycle.

In Figure 1, each circle represents one phase of software development life cycle. In each phase, the various components involved are represented in smaller circles For example, Requirements Elicitation, Requirements Analysis and Negotiation, Requirements Specification, Requirements Validation are the components of requirements phase(I). Design phase(II) includes components like Design system Architecture Design software components, Construct Design prototype, Design Validation. Various tasks like Arriving at Work break down structure Implementation using development tools , Unit Testing, Code validation are done at coding phase(III) and finally testing phase(IV) involves components like Test Analysis, Test plan/Test strategy, Test case components, Test Execution phase.

By incorporating the defect prevention process at each phase of software development, reduces the injection of defects at early stages itself thereby reducing the defect arrival rate as the project progress to the subsequent stages. The Defect prevention process includes four major steps like

(i) Collect Defect Data

(ii) Classify defect data using simple ODC classification scheme

(iii) Analyze the defect data for defect pattern/defect signature and

(iv) Suggest preventive actions in the form of Best Practices

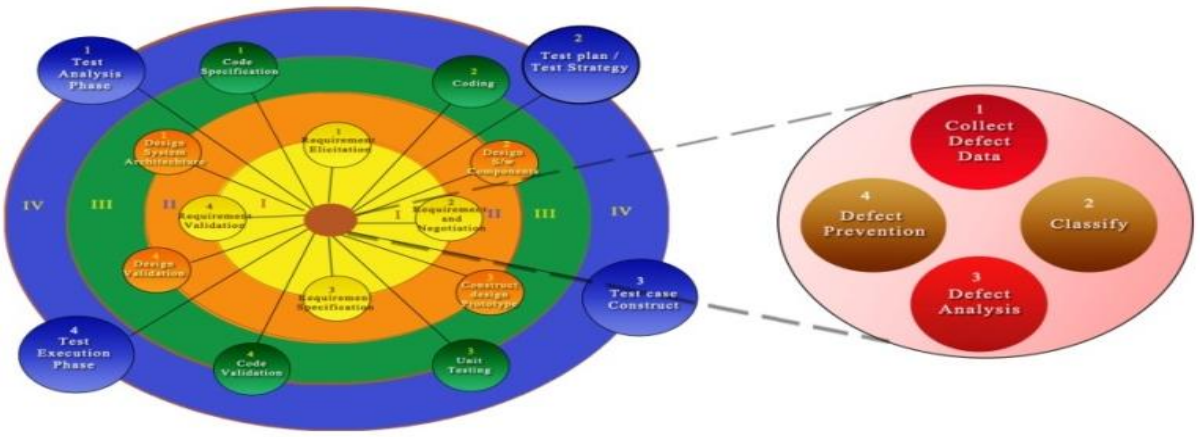

Fig 1: Defect Prevention Model

Applying ODC in software project

Orthogonal Defect Classification (ODC) is a technique used for the last few years in the industry to identify the root cause of the defects [9]. According to the Defect Prevention Model proposed in this study, the defect classification process is done through ODC. This work gives the details related to the actual steps involved in the classification of defects. According to ODC, when the defects are collected and analyzed in-process during an ongoing software development, information on defects is available at two specific points in time [9]. (1) When a defect is opened, the circumstances leading to the exposure of a defect and the likely impact to the user are typically known. (2)When a defect is closed after the fix is applied, the exact nature of the defect and the scope of the fix are known. ODC categories capture the semantics of a defect from these two perspectives. By defining the activities during a development process and their mapping to the ODC Triggers, an organization customizes the generic scheme to the local process.

ODC Defect Attributes

IBM's ODC classifies defect into eight defect attributes. Figure 2 depicts the ODC attributes used in this study for defect classification.

Activities in the opener section:

- **Activity** refers to the actual task that is involved (Inspection, Reviews, Testing etc.) when defects are found.
- **Trigger** describes the condition that had to exist for the defects to escape into subsequent phases.
- **Impact** relate to impact on users in terms of customer satisfaction.

Activities in the closer section:

- **Target** represents the high-level entity (i.e., design, code, ID, etc.) that was fixed.
- **Type** represents the nature of corrective action that was made on the defect.
- **Qualifier** captures the element of either nonexistent or wrong or irrelevant information.
- **Source** identifies the origin of the defect (Design, code etc.)
- **Age** identifies the history of the target (i.e., design, code, ID, etc.) that had the defect

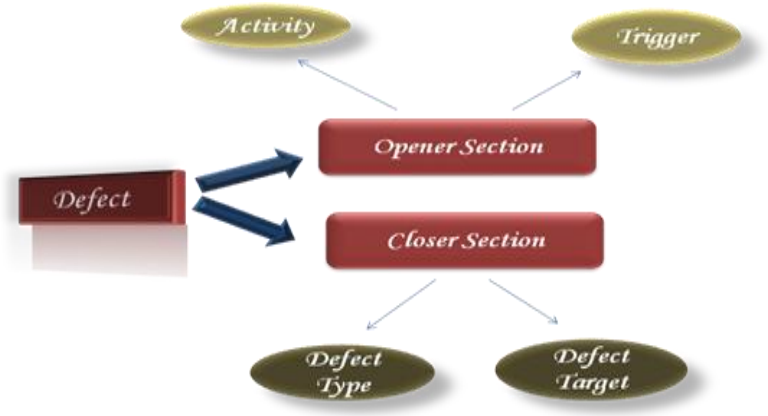

Fig 2: ODC defect attributes

Project Study

The implementation of defect prevention model is illustrated through an example of a commercial project executed at medium scale Software Company. The defect data were collected from a mini-ERP project. It is an 11 KLOC sized project which was done with a team of 20 members for a period of 4 months. The defect data were classified using ODC to understand the dynamics of defects. Based on the ODC classification, the semantics of defects were learnt and analysis of defect data was made to arrive at defect pattern (Fig 3 & 4). "Best Practices" were then arrived at in the form of defect prevention for the action team to implement in order to formulate process improvement

Activity/Defect Trigger

This study makes use of simple ODC classification scheme for categorization of defects. During the opener section, the major activities covered in this work are Design Review, GUI Review and Function Test. Defect trigger characterizes the process issues that allowed the defects to escape into later phases. For the projects taken for this study, Trigger for function test include Coverage, Sequence, Variation and Interaction and Trigger for GUI review includes Navigation, Input devices, Screen/Text characters, Widget/GUI behaviour and Widget/Icon appearance and the same has been depicted in Fig 3. For these triggers, the high level entity (Target) that has to be fixed include Requirement / Design and Coding in which majority fix of above 80% is attributable to coding phase alone.

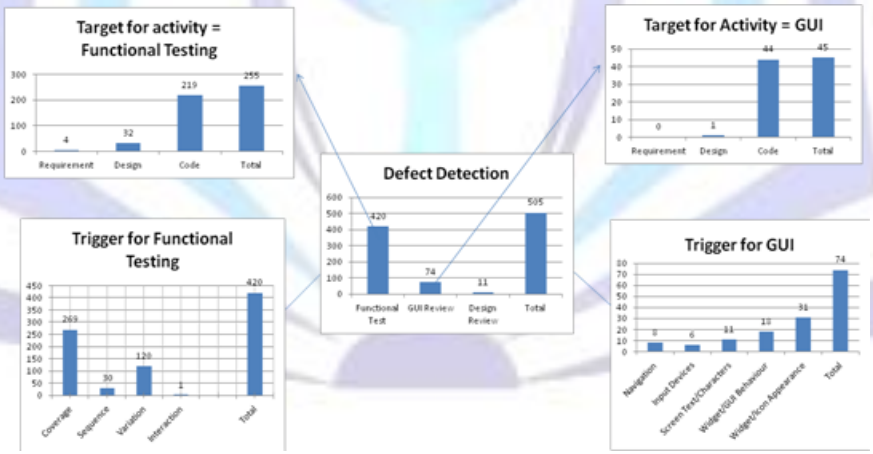

Fig 3: Simple Orthogonal Defect Classification

Defect Type

Defect type primarily deals with what caused the defect. A programmer making the correction usually chooses the defect type. In each defect type, a distinction is made between something *missing* or something *incorrect*. In this study, the five defect types identified were Algorithm, assignment, function, interface and checking. Figure 4 shows the defect types that affect coding and design of software development.

Algorithm: Defect due to problem in procedure or overloaded function.

Assignment: Defects due to values not initialized in few lines of code.

Function: Defect that affects end-user interfaces, product interfaces etc which requires the change in design.

Interface: Defect occur while interacting with other components or modules of the system

Checking: Defect in program logic which performs data validation check.

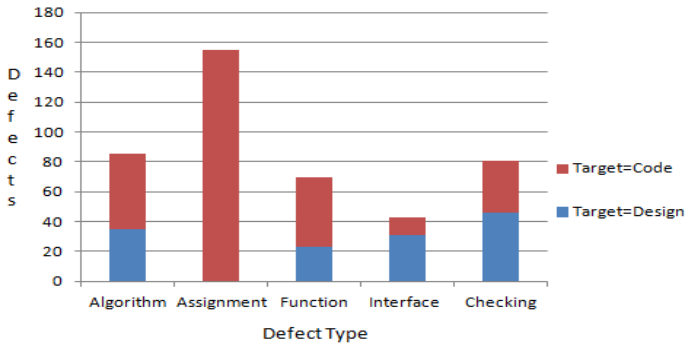

Figure 4: Defect Type distribution

Results and analysis

The following information was observed during the implementation of defect prevention model for Mini-ERP project. While the defect data were classified using ODC, It was found that most of the defects are related to Base Code (83%) and GUI (14%). Some observations were discovered when looking at Triggers of defect and compared to their Category. Figure 3 shows distribution of triggers and targets attributed to defect types represented from Functional testing and GUI. Analyzing Figure 4 shows how much of these defect targets are attributable to Algorithm, Assignment, Function, Interface and Checking of both coding and design phase. These observations are then analyzed to arrive at best practices approach for defect prevention to come out of these lacunae in the system based on ODC observations which has been tabulated in Table 1(Appendix). These best practices can be applied and further streamlined along with their leanings to make the development process cleaner and defect free. This will enable the company to have more focus on process and systems than on individuals.

CONCLUSION AND FUTURE ENHANCEMENT

To err is human, but defect prevention practices enhance the ability of software developers to learn from those errors and, more importantly, learn from the mistakes of others [13]. The benefits of implementing defect prevention are reducing overall cost, schedule, and resources and increasing the quality of a software product and the same is achieved through defect prevention model proposed in this study. The defect prevention model proposed in this study helps to eliminate the defects at every stage of software development, take preventive action for defect elimination and to avoid its recurrence. ODC way of classifying defects helped the practitioners point to the process area where preventive action has to be taken. This study made an attempt to deploy Simple ODC classification scheme into a project developed at medium scale IT industry, and paved the way to arrive at “Best Practices” to be followed for similar projects in order to realize the above benefits of defect prevention. This paper is limited to using some defect attributes of ODC for classification. As the job is human intensive requiring ODC trained personnel, planning to develop an open source tool which will automatically classify defect data based on ODC and generates a diagnostic report for taking preventive actions against the defect. When such a tool is developed, it will be of cost beneficial to small and medium scale IT industry and also help them to produce defect free IT solutions.

APPENDIX

Table 1: Best Practices To Be Followed - Preventive Measures Decided Based On Defect Data

Activity	Trigger Area	Defect Target	Best Practices To Be Adopted
Functional Testing	Coverage	Code	Most of the Functional defect is detected in the Code (85.88%) - so more attention in this area is essential for Defect Prevention.
			(a) Traceability Matrix, to trace each Functional Requirement till Source Code level should be made mandatory and Code to be released for Testing after a formal review of the updated Traceability Matrix for each requirement. An Independent review of the Traceability Matrix by the Quality Team should also be mandated.
			(b) Before commencing with the actual coding, Developers should document the Program Specification for each Source code and it should be formally reviewed by the Project Lead - This step would validate if the Developer has understood the actual requirement and can transform the Functional coverage in

Activity	Trigger Area	Defect Target	Best Practices To Be Adopted
			the Source code.
			(c) As the code related defects are categorized under Assignment, Algorithm/Method and Checking , which are more of a Source code related issues, the Developer writing the code should do a self code review of all possible criteria. Secondary code reviews by the peers or by the lead to be mandated to avoid such errors.
		Design	(a) Majority of design defect comes under Function/Class/Object Defect Type. The design perspective of each requirement should be understood and dependencies with external interfaces should be taken care. Formal Design review with key Technical members of the project, Interface teams, Vendors (if any), System and Database Administrators should be mandated before proceeding with the Coding phase.
			(b) The Technical Lead responsible for Design should have Overall knowledge about the Project Functionalities, Technical Implementation, System Environment, Deployment challenges, etc. to Design a perfect solution for the project. Any change in Design at a later stage would have a heavy impact in the Project timeline - so should be taken care in the Design phase itself.
		Requirement	(a) Requirement related defects, captured in the same phase is minimal - so can be ignored for further analysis.
	Variation	Code	(a) All negative scenarios and input of negative values should be handled in the Source code. So, developer should document all such negative cases in the Program Specification and it should be reviewed by the Technical Lead for completeness.
		Design	(a) A general list of negative cases to be handled should be included as a part of the Design document.
		Requirement	(a) In most cases, the negative scenarios to be handled are not documented as a part of the Functional Specification. It comes as a part of the experience and the Knowledge Base of the related project can be referred to avoid such errors.
	Sequence	Code	(a) Defects arising for the Trigger area 'Sequence', are more related to Integration related issues. It is just not enough for the Developer to understand his/her own code related functionality, but a knowledge on the overall project is essential to avoid such defects. Shared source code / common routine in the project has to be discussed, documented in detail and should be agreed between the stakeholders involved in Integration. A final approved document on common routines is a must to avoid such errors (To be circulated before Source code development).
		Design	(a) The document specified above on Workflow Sequence should be written and agreed in the Design phase.
		Requirement	(a) Scenarios that may have a sequential execution must be well documented with sample Use Cases.
	Interaction	Code / Design / Req	(a) This is related more to the 'Sequence' Trigger area, as it involves the error that arrises in the sequence of execution. So, Best practices specified in Sequence Trigger Area can be referred here.
GUI Review (14.65%)	Widget/ Icon Appearance	Code	Most of the GUI Review commnets is in the Code (97.78%) - so more attention in this area is essential for Defect Prevention.

Activity	Trigger Area	Defect Target	Best Practices To Be Adopted
			(a) Before starting the actual coding, a prototype of the application should be built to finalize on the look and feel of the application.
		Design	(a) Screens of the prototype can be embedded in the Design document to understand the application better. Senior Technical developer along with the team can work in parallel with the Technical lead to assist him in bringing such add on features in the Design document.
	Widget/ GUI Behaviour Navigation	Code / Design	(a) Straightforward scenarios of the GUI Behaviour can be captured in the prototype. Other scenarios on Page reload, browser compatibility, etc can be referred from the Knowledge Base of the similar successfully implemented projects and this has to be documented in the Design document for the project.
	Screen Text/ Char	Code / Design	(a) Font, Size, Colour and other look and feel features can be broughtout upfront if all this are taken care in the Prototype Development for the project. Cascade Style Sheets (CSS) can be developed and used in common among all Developers of the project.
	Input Devices	Code / Design	(a) As a part of the Requirement capture, details on the Input Devices to be supported should be well documented. Program Language support for various Input devices has to be analyzed and documented in the Design Manual.
System Test (2.18%)			(a) System Test related defects is minimal - so can be ignored for further analysis.

REFERENCES

[1] Suma V, T V Rajagopalakrishnan Nair, “Defect Prevention Approach in Medium Scale IT Enterprises”, National Conference on Recent Research Trends in Information Technology” Banglore (2008)

[2] Karthikeyan Natesan, “Using Defect Prevention Techniques in SDLC”, “International Journal of Information Technology & Computer Science (IJITCS) (ISSN No : 2091-1610) Volume 5 : Issue on September / October , 2012

[3] Prakriti Trivedi, Som Pachori, "Modelling and Analysing of Software Defect Prevention Using ODC", “(IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 1, No. 3, September 2010

[4] Marc McDonald; Robert Musson; Ross Smith, “The Practical Guide to Defect Prevention”, Microsoft Press, 2007.

[5] Norm Bridge, Corinne Miller “Orthogonal Defect Classification Using Defect Data to Improve Software Development”, Motorola Corporate Software Center, Schaumburg, Illinois.

[6] Muhammad Faizan,Muhammad Naeem Ahmed Khan,Sami Ulhaq, “Contemporary Trends in Defect Prevention: A Survey Report”, International Journal of Modern Education and Computer Science (IJMECS) april 2012

[7] Pankaj Jalote, K. Dinesh, S. Raghavan, M. R. Bhashyam, M. Ramakrishnan “Quantitative Quality Management through Defect Prediction and Statistical Process Control”, Infosys Technologies Ltd. (2000)

[8] Khaleel Ahmad, Nitasha Varshney, “On Minimizing Software Defects during New Product Development Using Enhanced Preventive Approach” International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-5, November 2012

[9] Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., and Wong,

[10] M.-Y., "Orthogonal Defect Classification - A Concept for In-Process Measurements," IEEE Transactions on Software Engineering, vol. 18, no. 11, November 1992, pp.943-956.

[11] Yang GU “Adopting ODC to improve software quality: A case study” (2006)

[12] http://www.quotationspage.com/quote/5219.html

[13] Mukesh Soni, “Defect Prevention: Reducing Costs and Enhancing Quality” (2010)

ISSN 2277-3061

Author' biography with Photo

[1]First Author – Sakthi Kumaresh is a Ph D candidate at Bharathiar University, Coimbatore, India; Currently working as

Associate professor at Department of Computer Science, MOP Vaishnav College, Chennai. She obtained her Master's Degree from Madurai Kamaraj University, Madurai, TN, India in 1996 and M Phil in Computer Science from Periyar University, Salem, TN, India in 2006. She has decade of teaching experience. Her areas of specialization include Software Engineering, Software Project Management, Software Testing, Software Quality Management and Unified Modeling Language. She is doing research in the area of Software Quality Engineering. She has publications in National conferences and International journal..

[2]Second Author – Dr. Baskaran Ramachandran is working as the Associate professor in Department of computer science, Anna University, Chennai. He has obtained his M.E. and Ph.D. in the field of Computer Science and Engineering in Anna University, Chennai, India. He is having around a decade of experience as an Academician and his research areas include Multimedia and principles, Software quality engineering, Software Agents and Distributed networking. He has published around 50 research papers in National and International Journals and Conferences. He is the member of various forums. He is the editor and the reviewer in various journals. He is guiding research scholars working in area of software standards for Attributes Specific SDLC Models & Evaluation and Metric Based Efficient Traffic Management.