Defect Analysis and Prevention for Software Process Quality Improvement

Sakthi Kumaresh
Research Scholar, Bharathiar University.
Department of Computer Science,
MOP Vaishnav College for Women, Chennai.

R Baskaran
Asst. Professor,
Dept. of Computer Science & Engg
Anna University, Chennai.

ABSTRACT

"An ounce of prevention is worth a pound of cure." In software, these expressions translate into the common observation that the longer a defect stays in process, the more expensive it is to fix [10]. Moreover software defects are expensive and time consuming. The cost of finding and correcting defects represents one of the most expensive software development activities. And that too, if the errors get carried away till the final acceptance testing stage of the project life cycle, then the project is at a greater risk in terms of its Time and Cost factors. A small amount of effort spent on quality assurance will see good amount of cost savings in terms of detecting and eliminating the defects.

To gain a deeper understanding of the effectiveness of the software process, it is essential to examine the details of defects detected in the past projects and to study how the same can be eliminated due to process improvements and newer methodologies. This paper will focus on finding the total number of defects that has occurred in the software development process for five similar projects and aims at classifying various defects using first level of Orthogonal Defect Classification (ODC), finding root causes of the defects and use the learning of the projects as preventive ideas. The paper also showcases on how the preventive ideas are implemented in a new set of projects resulting in the reduction of the number of similar defects.

Keywords

Defect, Defect Analysis, Defect Prevention, Root Cause Analysis

1. INTRODUCTION

Software Defect can be defined as "Imperfections in software development process that would cause software to fail to meet the desired expectations".

In software development, lot of defects would emerge during the development process. It is a fallacy to believe that defects get injected in the beginning of the cycle and are removed through the rest of the development process [8]. Defects occur all the way through the development process. Hence, defect prevention becomes an essential part of software process quality improvement.

Defect prevention (DP) is a process of improving quality whose purpose is to identify the common causes of defects, and change the relevant process(es) to prevent that type of defect from recurring[2]. DP also increases the quality of a software product while reducing overall costs, schedule and resources. This ensures a project can maintain cost – schedule – quality equilibrium.

The purpose of defect prevention is to identify those defects in the beginning of the life cycle and prevent them from recurring so that the defect may not surface again. In this study, in order to improve software process quality, defects are first identified from a given set of projects, classified and analyzed for patterns. These patterns are then eliminated by finding the root causes, for which preventive mechanisms are established for reducing re-occurrences of similar defects in the subsequent projects, thus improving Quality. This Cycle will be continuous to improve Quality of the SDLC. The scope of this paper is to provide a comprehensive view on the defect prevention techniques and practices that can be followed in software development.

The rest of the paper is organized as follows: Section 2 presents an overview of related work. Section 3 discusses the need for defect prevention. Section 4, presents the process improvement workflow along with the illustration of various stages. The distribution of the project defect data across project is illustrated in section 5. Section 6 presents the root cause analysis and determination of preventive action. Section 7 displays the reduction of defects in a new project that inherited the preventive ideas from old projects. Finally, in Section 8, the paper is concluded by highlighting the benefits of adopting preventive action in the subsequent project thereby improving the software process quality.

2. RELATED WORK

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 eliminated. With the advent of SDLC processes many companies formulated their own defect prevention mechanisms and many studies were conducted towards defect prediction and prevention.

One study by Fang Chenbin [6] was introduction of a tool called Bug Tracing System (BTS) for defect tracing, has the advantage of popularity and low cost, and also improves the accuracy of tracking the identified defects. Work done by Stefan Wagner [9] summarizes the work on defect classification approaches that have been proposed by two companies IBM and HP. The IBM approach is called Orthogonal Defect Classification (ODC) and the HP approach is based on three dimensions -Defect Origin, Types and Modes. Pankaj Jalote and Naresh Agarwal [7] stressed on how analysis of defects found in first iteration can provide feedback for defect prevention in later iterations, leading to quality and

productivity improvement. Ajit Ashok Shenvi [1] worked under the philosophy that "capturing defects in the earlier stage of the life cycle" is a means of preventing defects in the later stages of the product life cycle and concentrated on finding out preventive action for functional defect types only. Suma V [11] aimed to provide information on various methods and practices supporting defect detection and prevention based on three case studies and studied about the defect detection and defect prevention strategies adopted in these three projects only. 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 ODC, 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.

3. NEED FOR DEFECT PREVENTION

Defect prevention is an important activity in any software project. 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. It also enhances the total productivity.

4. PROCESS IMPROVEMENT WORK FLOW

Figure 1 Process Improvement Workflow

4.1 WORK FLOW STAGES

4.1.1 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.

4.1.2 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 [8]. 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.

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 code using functional/unit testing), and Documentation. Defects are classified under these types and then analysis of defects is carried out.

4.1.3 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 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 [8].

4.1.4 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 [4].

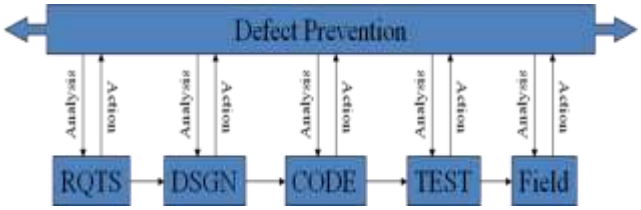

Figure 2: Defect Prevention in Software Lifecycle

4.1.5 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.

5. PROJECT DEFECT DATA

To study the prevalence of defect in software development process, five projects are identified. Specifically, these selected projects were developed under **Microsoft .net platform**. 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. Defect density is a measure of the total number of defects in a project divided by the size of the software being measured [3].

Defect Density (DD) = Number of defects / size (kloc) – (1)

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 not follow any preventive action.

Table 1: Defect Density

First set of Project Proj No	Proj Name & Description	KLOC	No of Defects	Effort (P Hr)	Defect Density
Proj 1	Maple - CRM module for a trading company	29	172	2200	0.006
Proj 2	Indesign – Survey Automation Tool	14	119	1200	0.009
Proj 3	Stock Market Application	7	104	600	0.015
Proj 4	Issue Tracker- Manages and maintains list of issues raised by an organization	5	97	400	0.019
Proj 5	GRTNET - General Reporting Tool	31	145	2400	0.005

Figure 3: Defect - Size Correlation

The project size can be measured either in terms of kilo lines of code (KLOC) produced or in terms of Function Point (FP). For the projects that are taken for study, the project size is measured in terms of KLOC. Comparison is then made between KLOC and number of defect produced by the project. This comparison is depicted in the above figure. From (fig 3), it is evident that, the number of defects in the project varies as the size of the project varies.

Table 2: Categorization of defects across phases for five similar projects

Life cycle phases	Activity	Defect Type	No of Defects
Requirements	Review	REQ	74
Design	Review	DSN	58
Code	Testing (Function/unit)	LOG	420
GUI	Review	GUI	55
Documentation	Review	TYP	30

Table 3: Code Description

Code	Name	Description of Defect Type
LOG	Logical Error	Logical Error
REQ	Requirements	Error in understanding the requirements, or inadequate requirements definition.
GUI	Graphical Error	Error in screen/report layout and design
DSN	Design Error	Error in developing design, or inadequate design, or technical inadequacy in design.
TYP	Documentation	Typographical error in documentation or in code, including spelling errors, mistyped words, and missing delimiters in code.

Table 4: Observed defect pattern across projects

Proj No	REQ	Design	LOG	GUI	Doc	Total
Proj 1	20	17	120	9	6	172
Proj 2	15	11	75	10	8	119
Proj 3	12	14	58	13	7	104
Proj 4	12	8	60	15	2	97
Proj 5	15	8	107	8	7	145
Total	74	58	420	55	30	637

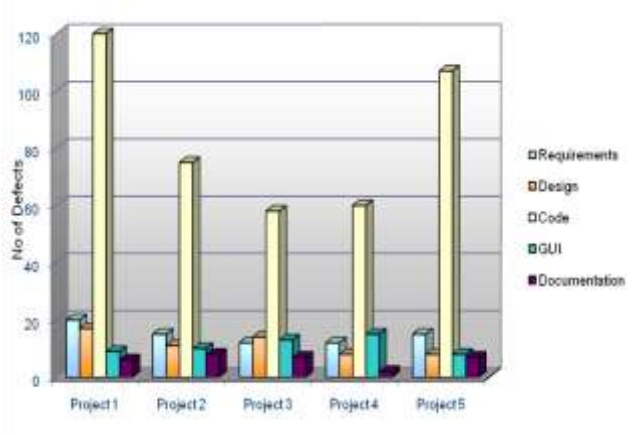

Figure 4: Defect Type pattern across project

Fig 4 illustrates the defect type pattern for five similar projects that are shown in Table 1. It is found that 70-80 percentages of defects were classified as coding defects. Approximately 10% of defects are GUI defects. Balance 10% defects are Requirements, Design and Documentation defects.

6. DEFECT ANALYSIS

6.1 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. Pareto chart for various defect types of the projects mentioned in table 4 is shown in Figure 5.

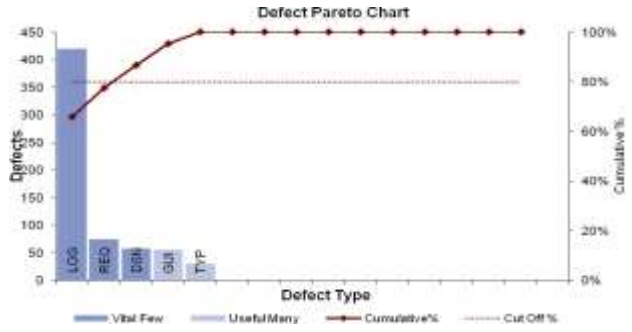

Figure 5: Defect Pareto chart

The Pareto Chart shows the frequencies of occurrences of the various categories of problems encountered, in order to determine which of the existing problems occur most frequently. The problem categories or causes are shown on the x-axis of the bar graph and the cumulative percentage is shown on the y axis of the graph. From the Pareto chart, it is understood that 80% of the defect are falling under the category Logical, Requirement, Design defect type. These defect types should be given higher priority and must be attended first.

6.2 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. For the projects mentioned in Table 1, the major causes making software defect to happen are represented using a cause-and-effect diagram, as shown in Figure 6.

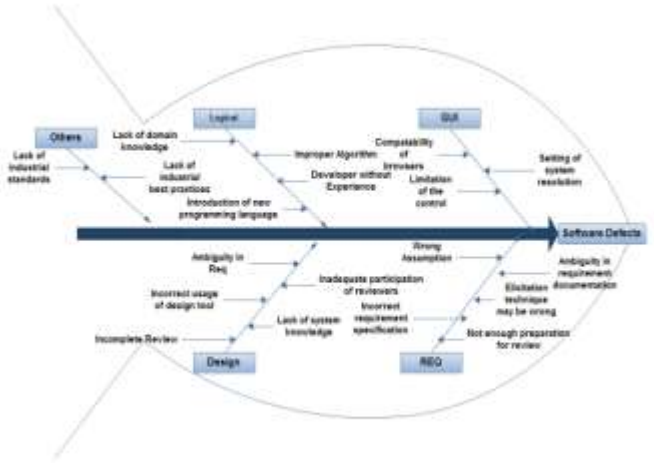

Figure 6: Cause Effect Diagram for a Software Defect

6.3 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.

Table 5 shows possible preventive actions that were agreed by the project team members for the various root causes of the defect.

Table 5: Root causes and preventive action

S.No	Defect Type	Root Cause	Preventive Actions
1	LOG	1) Lack of Domain knowledge. 2)Improper Algorithm 3) Developer without experience 4) Introduction of new programming language	• Domain knowledge: Training should be given to the team members before starting the next phase or a new project. The Training Calendar and the trainings attended by the team can be tracked as a part of the Project status review meeting. • Make available of trained and experienced resources for coding and testing. Plan for trained resources well in advance and if they are not available train the existing resources. Any risk on the availability of the trained resources should be tracked as a part of the Risk Management Worksheet. • Generally introduction of new programming language should be known well in advance to the team and proper training should be given well in advance.
2	REQ	1) Assumption of the Requirement gathering person in the grey Area. 2) Ambiguity in requirement documentation 3)Incorrect requirement specification 4)Wrong elicitation technique 5)Not enough preparation for review by reviewers	• Discuss more about the boundary of the applications and granularity of each requirement • Using Business Analysts /Domain professionals during requirement elicitation. • Requirement workshop (For clarity & common understanding of implicit & explicit requirements with all teams including testing) • Frequent communications with customer will help to know his requirements • A formal sign off from all Business Users who would handle the application should be mandated before starting the design phase.
3	DSN	1)Ambiguity in requirement documentation 2)Incorrect usage of design tool 3)Incomplete review 4)Inadequate participation of reviewers 5)Lack of system knowledge	• Discuss more about the boundary of the applications and granularity of the requirement. The equivalent design conversion should be well documented in the Design Document and sign off should be received before starting the coding. • Care should be taken in choosing right tool • Training should be given in the usage of design tool • The design document should be consistent with requirements specification. The review should be carried out with a Design review list as base and adequacy in review should be cross checked by the Quality team or Organisation Design review team.
4	GUI	1) Compatibility of browsers, supporting S/W, H/W etc. 2)Settings of the system Resolution, 3)Limitations of the Control	• Most of the Graphical defects appear similar across all projects. Maintain a defect database and run test cases through it before starting up with the project
5.	TYP	Oversight	• A thorough check shall be done before delivering the artifact. • Customer review of artifacts and deliverables

7 IMPLEMENTATION OF DEFECT PREVENTIVE (DP) ACTION

To see the effectiveness of using the DP action, the above mentioned preventive action are implemented in the next set of five similar projects, and the process improvement was observed in terms of average defect density.

Table 6: Defect Density after implementing Defect Preventive action

Second set of Project Proj. No	Proj Name & Description	KLOC	No of Defects	Efforts (P Hr)	Defect Density
Proj. 1	eCampus HR Module	39	118	2900	0.003
Proj. 2	Content Management System - Additional Reporting	11	54	925	0.005
Proj. 3	MOSS based document management system	16	152	1950	0.010
Proj. 4	UAE Finance module development	9	97	1290	0.011
Proj. 5	R&D based BI tool	33	267	2450	0.008

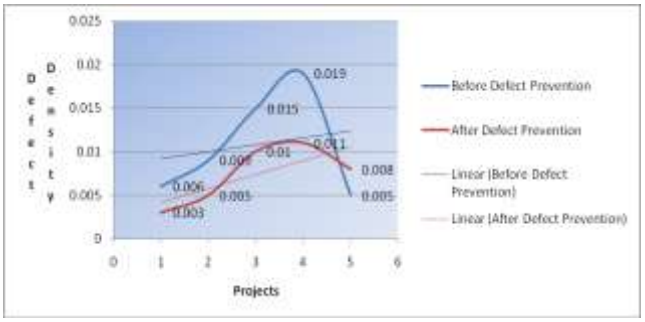

Figure 7: Graph depicting Defect density comparison before and after implementation of preventive action

The Graph represents the distribution of defect densities for 5 similar projects before and after implementing the Defect Prevention as provided in the Table 1 and Table 6. Trend line

shows that the defect density after implementing DP is well below that of defect density before the DP implementation.

The average defect density has gone down from 0.0108 (first set of projects-Table 1) to 0.0074 (second set of project –Table 6). By implementing the defect preventive action, not only reduces the defect density, rework effort is also reduced due to which effort involved in various processes is also reduced considerably.

8. CONCLUSION

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 study 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.

9. REFERENCES

[1]. Ajit Ashok Shenvi,2009,"Defect Prevention with Orthogonal Defect Classification", In Proc- ISEC '09, February 23-26, 2009

[2]. Defect Prevention by SEI's CMM Model Version 1.1., http://www.dfs.mil/technology/pal/cmm/vl/dp

[3]. Linda Westfall, Defect Density http://www.westfallteam.com/Papers/defect_density.pdf

[4]. Megan Graham, 2005, Software Defect Prevention using Orthogonal Defect Prevention. http://twin-spin.cs.umn.edufiles/ODC_TwinSPIN

[5]. Mukesh soni, 1997, Defect Prevention: Reducing cost and improving quality" published in IEEE Computer, (Volume 30, Issue **8)**, August 1997.

[6]. Pan Tiejun, Zheng Leina, Fang Chengbin, 2008, "Defect Tracing System Based on Orthogonal Defect Classification" published in Computer Engineering and Applications, vol 43, PP 9-10, May 2008.

[7]. Pankaj Jalote, Naresh Agarwal, 2007, "Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development" In proc- ITI 3rd International Conference on Information and Communications Technology, pp. 703-713.

[8]. Ram Chillarege, Inderpal S Bhandari, Jarir K Chaar, Michael J Halliday, Diane S Moebus, Bonnie K Ray, Man-Yuen Wong, 1992, "Orthogonal Defect Classification - A Concept for In-Process Measurements", IEEE Transactions on Software Engineering, Vol. 18, No.11, Nov 1992. http://www.chillarege.com/odc/articles/odcconcept/odc.html

[9]. Stefan Wagner, 2008,"Defect Classification and Defect Type Revisited" Proceedings of the 2008 workshop on Defects in large software systems, (DEFECTS'08) pages 73-83, ACM Press,2008

[10]. Steve McConnel, "An ounce of prevention", IEEE software, May/June 2001

[11]. Suma V and T R Gopalakrishnan Nair , 2008, " Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels" Proceedings of World Academy of Science, Engineering and Technology Volume 32 August 2008