ISSN: 2226-7522(Print) and 2305-3327 (Online)
Science, Technology and Arts Research Journal
Jan-Mar 2013, 2(1): 64-68
www.starjournal.org

Original Research

Software Excellence Augmentation through Defect Analysis and Avoidance

Abdul Kadir Khan

[1]College of Computing and Informatics, Post Box No: 337, Haramaya University, Dire Dawa, Ethiopia

Abstract

The key challenge for any software organization is to develop a software product with less post deployment defects. Moreover if the defects reach till the deployment face then the project will be at a higher risk in terms of its cost and time aspects. A small amount of initial effort on software quality will definitely save a good amount of cost and time compare to defect recognition and removal strategy. This paper will focus on finding the total number of defects that has occurred in the SDLC (software development life cycle) for five similar type of projects done by final year graduating students in Haramaya University. The technology used in all the five projects is also same. Orthogonal Defect Classification (ODC) is the most prevailing technique for identifying defects wherein defects are grouped into types rather than measured independently. This technique highlights those areas in SDLC that require attention. The paper will also focus on finding root causes of the defects and use the learning of the projects as preventive ideas. In the remaining part of the paper, the preventive ideas are used in a new set of projects developed by the graduating students in next phase resulting in the reduction of the number of similar defects.

Article Information

Article History:
Received : 18-01-2013
Revised : 22-03-2013
Accepted : 26-03-2013

Keywords:
Defect Prevention
Software Quality
Orthogonal
Defect Classification
Defects

***Corresponding Author:**
Abdul Kadir Khan

E-mail:
qadirforu@gmail.com

INTRODUCTION

Defect prevention is a group task. When an IT organization adopts a defect prevention strategy, it keeps analyzing and acting on defect data directly with its development group members. It exactly tells the cost of group's errors and challenges the group to avoid the defects, by taking the responsibility of product quality. This practice results in giving economical high quality design and motivates the collection of high quality dimensions. It creates a competent system where feedback is continually used to optimize the design and check up processes.

In developing a project, a 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 (Paulk, 1993). Defects happen all the way through the development process. Hence, defect prevention becomes a critical part of software process quality improvement.

The most effective way to manage defects is to prevent their initial introduction. In the PSP, there are three different but mutually supportive ways to prevent defects. The first is to have engineer's record data on each defect they find and fix. Then they review these data to determine what caused the defects and to make process changes to eliminate these causes. By measuring their defects, engineers are more aware of their mistakes, they are more sensitive to their consequences, and they have the data needed to avoid making the same mistakes in the future. The rapid initial decline in total defects during the first few PSP course programs indicates the effectiveness of this prevention method.

The second prevention approach is to use an effective design method and notation to produce complete designs. To completely record a design, engineers must thoroughly understand it. This not only produces better designs; it results in fewer design mistakes.

The third defect prevention method is a direct consequence of the second: with a more thorough design, coding time is reduced, thus reducing defect injection. PSP data for 298 experienced engineers show the potential quality impact of a good design. These data show that during design, engineers inject an average of 1.76 defects per hour, while during coding they inject 4.20 defects per hour. Since it takes less time to code a completely documented design, by producing a thorough design, engineers will correspondingly reduce their coding time. So, by producing thorough designs, engineers actually inject fewer coding defects (Watts, 2000).

In the Figure 1, Software Production, Software Testing, Problem Database blocks and processes associated with these blocks symbolize the handling of defects within the existing philosophy of most of the software organizations.

The blocks (Cause Analysis Meeting, Action Team Meeting) and the processes associated with these blocks represent the important part of the defect prevention methodology. The vital process of the defect prevention methodology is to examine defects to get their root causes, to decide a quick solution and preventive action. These preventive procedures, after consent and commitments from team members, are embedded into the organization as a baseline for future projects. The methodology is aimed at providing the organization a long-term solution and the maturity to learn from mistakes.

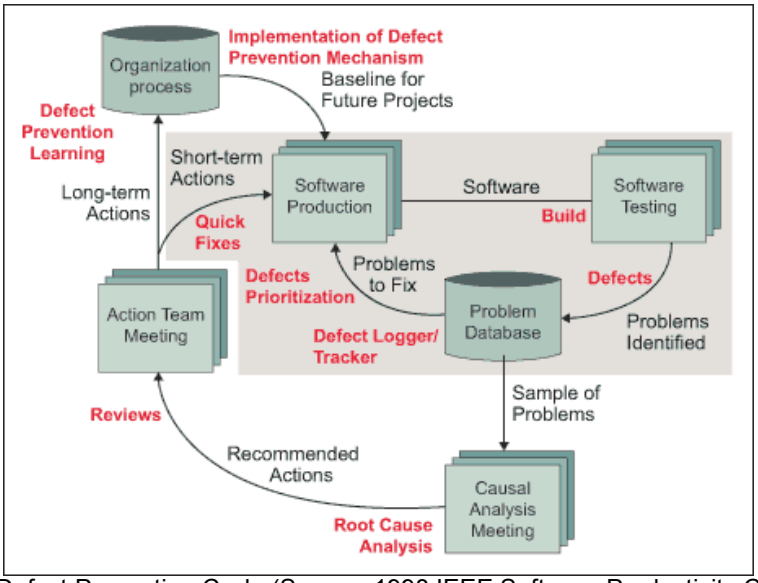

Figure 1: Defect Prevention Cycle (Source: 1998 IEEE Software Productivity Consortium.

Most of the activities of the defect prevention methodology require a facilitator. Since final year projects developed by the graduating students represent the real life project and are very similar to the projects taken by different commercial organization, a facilitator was chosen from every team to facilitate the research methodology. Five similar types of projects were selected to carry out this research. All the groups used the same technology (PHP with SQL Server) to carry out the work. For each group, an advisor/Technical Expert from the college is assigned to help the students time to time. The facilitator is involved in arranging meetings, communication with advisor/technical experts, and consolidating the defect prevention guidelines. Many software organizations have designed their own defect prevention methodologies in the beginning of SDLC processes. So many researchers have also conducted their own independent research to predict and prevent the defects. A tool named Bug Tracing System (BTS) for defect tracing was introduced by Fang Chenbin (2008). It got popularity because of its low cost and defect tracking accuracy. IBM and HP, the two well known companies in software field, have their own defect classification approach. IBM approach is known as Orthogonal Defect Classification (ODC) and the HP approach is based on three dimensions -Defect Origin, Types and Modes. Pankaj Jalote and Naresh Agarwal (2007) stressed out on how analysis of defects originated in first iteration can provide view for defect prevention in later iterations, leading to quality and advent output improvement. In this paper, through the combination of above research findings, a better defect prevention cycle has been introduced for the better software quality.

Steps for Improving Quality

Projects Data Summary

Information like kilo number of lines of code (KLOC) produced by the software and number of defects in the project are collected. Then Defect density is measured to track the impact of defect reduction and to judge the quality improvement.

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

1 KLOC = 1000 lines of codes- (2)

Table 1: 1st Set of Projects Data Summary.

Project No.	Name of the Project	KLOC	No. of Defects	Defect Density
1	Haramaya University Human Resource Management System	7	104	0.014
2	Metrology Agency Automation System in Oromia	5	97	0.019
3	Clearance System for Haramaya University	9	150	0.016
4	Haramaya University Students Dormitory Placement System	11	93	0.008
5	Harari Region Tourism Guide and Information System	6	79	0.013

Average Defect Density in Table I = 0.014.

Table 2: Categorizing Defects.

SDLC Stage	Defect Type	Activity	No. of Defects
Requirement & Analysis	RQA	Review	72
Design	DSN	Review	148
Coding	LOG	Testing	180
GUI	GUI	Review	40
User Manual	DOC	Review	83

Table 3: Defects Detail and descriptions.

Defect Type	Full Name	Defect Description
RQA	Requirements Error	Insufficient requirement Definition
DSN	Design Error	Inadequate design or faulty design
LOG	Logical Error	Error in Programming or faulty code
GUI	Graphical Error	Report layout errors
DOC	Typing Error	Spelling mistakes or mistyped code

Table 4: Categorizing Defects across Projects.

Project No.	Defect Type					Total
	RQA	DSN	LOG	GUI	DOC	
1	17	20	35	10	22	104
2	13	24	30	15	15	97
3	19	31	59	11	30	150
4	9	29	43	2	10	93
5	13	44	13	2	6	79

Research Findings

From the above tables, the research outcomes are given below:

- 34.41% of total defects were found in coding stage (Table 2).
- 40.06% of total defects were found in requirements and design stages (Table 2).
- 15.86% of total defects were found in documentation part which is very unusual if we look at the software defects globally.
- In Ethiopia, Students does not have a good command over English Language.
- GUI stage has least number of defects.

Root Cause Analysis

It is a way of finding activities/processes which causes errors/defects and also finds out the activities/processes to reduce the defects by providing remedial measures. Root cause analysis works on 2 main principles:

i. Self Review – The developer himself improves the software quality by revisiting the SDLC stages where defects/errors were found.
ii. Peer Review –Objective of peer as well as self review is same (i.e. to remove the defects). In peer review, local or third party technical expertises are called to remove the defects.

Cause and defect diagram is used to know the causes which generate defects. It is also known as fishbone diagram which is commonly used for knowing the defect causes. A simple cause–defect diagram is shown in Figure 2.

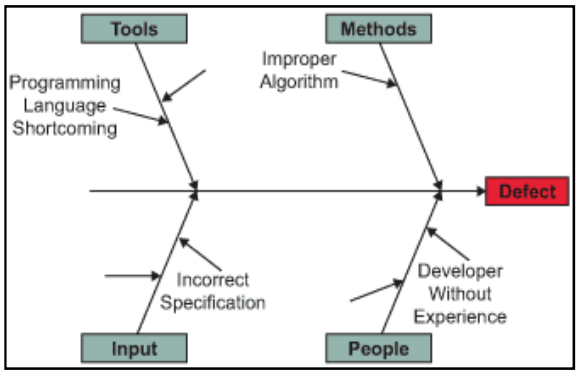

Figure 2: Cause and effect diagram for a defect.

This diagram is generally developed in a brainstorming session by the working team. Once the causes are listed, elimination methods require another brainstorming session.

Table 4 shows the result of brainstorming session for root causes of defects and possible preventive plans by another brainstorming session.

Table 5: Root Causes and Preventive Plan.

Defect	Root Cause	Preventive Plan
RQA	▪ Lacking clarity in requirement documentation. ▪ Less preparation by reviewers. ▪ Incorrect way of data gathering. ▪ Taking the stage very lightly	▪ Arranging requirement meetings to know the exact information. ▪ Reviewing the result with full attention. ▪ Using professionally approved way for data gathering. ▪ Continuously in touch with client for exact information gathering.
DSN	▪ Improper use of designing tool. ▪ Inadequate requirement information ▪ Insufficient system knowledge. ▪ Improper review	▪ Select the proper design tool. ▪ Proper System knowledge through different meetings/workshop. ▪ Cross checking the reviewed document. ▪ Good knowledge of System design. ▪ Proper training of design software
LOG	▪ Technically weak ▪ Lack of new technology/language knowledge ▪ Improper algorithm. ▪ Lack of experienced staffs	▪ In advance, proper training should be provided for the new languages/technology. ▪ Experienced and technically sound staffs should be hired. ▪ Different assessment methods should be implemented to know the skills of staffs.
GUI	▪ Software is incompatible with hardware or vice versa. ▪ Improper system settings. ▪ Lack of advanced graphic applications.	▪ Knowledge of hardware requirements in advance. ▪ Installation of latest graphics applications. ▪ Manage the system resources properly.
DOC	▪ Not a good command over English language. ▪ Grammatically incorrect sentences.	▪ Organise workshops to strength the English language. ▪ Proper review of the documentation before release

Effects of Implementing the Defect Preventive Plan

To see the effects of preventive plan discussed in table 5, same group of students were given five new similar types of projects to work on.

Table 6: 2nd Set of Projects Data Summary (after Implementing Table 5 Preventive Plans)

Project No.	Name of the Project	KLOC	No. of Defects	Defect Density
1	Haramaya University Model School Management System	8	76	0.009
2	Haramaya University Students Cafeteria System	5	43	0.008
3	Haramaya University Job Placement System	10	113	0.011
4	Haramaya University Finance Management System	7	49	0.007
5	Movie on Demand Website	9	91	0.010

Average Defect Density from Table 6 = 0.009.

It is quite clear that the average defect density (0.009) of table VI is less than the average defect density (0.014) of table I. So after implementing the preventive plans, the numbers of defects in second set of similar projects are less.

CONCLUSION

Implementation of defect preventive strategies helps in getting quality product. It is also a good way to learn from our own mistakes as well as from others' mistakes. Defect preventive plans help us in Reducing cost and time without compromising the product quality. It reduces rework and makes better understanding among the team members. It also creates better professional environment through helping each others. It increases the customer satisfaction and productivity. Through Defect prevention strategies, we also creates better teaching learning environment and avoids the same mistakes to be happen again. This research also confirms that Orthogonal Defect Classification (ODC) approach accepted by IBM is very useful to get better knowledge about defects. Through ODC, we get better ideas to avoid the errors or defects in the future projects. Defect preventive strategy derived from ODC help us in finding the solutions to our common mistakes in software development process.

REFERENCES

Adam A. Porter., Carol A. Toman and Lawrence G. Votta. (1997). An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development. *IEEE Transactions on Software Engineering*, 23 (6).

Chillarege, I.S., Bhandari, J.K., Chaar, M.J., Halliday, D.S., Moebus, B.K., Ray M., Wong, Y. (1992). "Orthogonal Defect Classification-A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering*, 18 (11), 943-956.

Chillarege. (2002)."Test and development process retrospective a case study using ODC triggers".

Jasmine, K.S., Vasantha, R. (2007). "DRE – A Quality Metric for Component based Software Products", proceedings of World Academy Of Science, *Engineering and Technology*, 23.

Pan Tiejun, Zheng Leina, Fang Chengbin. (2008). "Defect Tracing System Based on Orthogonal Defect Classification". *Engineering & Applications*, 43, 9-10.

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

Paulk. (1993). "Capability Maturity Model for Software" Version 1.1, Mark C.Paulk, Bill Curtis, Mary Beth Chrissis, Charles V.Weber, Software Engineering Institute.

Bonnie K Ray, Man-Yuen Wong. (1992). "Orthogonal Defect Classification - A Concept for In-Process Measurements", *IEEE Transactions on Software Engineering*, 18(11).

Steve McConnel. (2001). "An ounce of prevention", IEEE software.

Watts S. Humphrey. (2000). "The Personal Software Process (PSP)".

Yang. (1992). "Orthogonal Defect Classification A Concept for In- Process Measurements"