Defect Prevention Technique in Test Case of Software Process for Quality Improvement

ABSTRACT
A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle. Defect prevention is the most vital but habitually neglected facet of software quality assurance in any project. If functional at all stages of software development, it can condense the time, overheads and wherewithal entailed to engineer a high quality product. The key challenge of an IT industry is to engineer a software product with minimum post deployment defects. This paper will focus on finding the total number of defects if the test case shows that the software process not working properly. That has occurred in the software development process. For three similar projects and aims at classifying various defects using first level of Orthogonal Defect Classification (ODC), finding base causes of the defects and uses 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.

1. KEYWORDS
Test case, Defect, Defect Analysis, Defect Prevention, Root Cause Analysis

2. INTRODUCTION
Test cases are the specific inputs that you'll try and the procedures that you'll follow when you test the software. A sequence of one or more subtests executed as a sequence because the outcome and/or final state of one subtest is the input and/or initial state of the next. The word ‘test’ is used to include subtests, tests proper, and test suites. 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.

3. WHY DEFECTS 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

4.1 WORK FLOW STAGES

4.1.1 Test Case
A test case has components that describe an input, action or event and an expected response, to determine if a feature of an application is working correctly."

4.1.2 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.

4.1.3 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.

4.1.4 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].

4.1.5 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].

4.1.6 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, three 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].

$$\text{Defect Density (DD)} = \frac{\text{Number of defects}}{\text{size (kloc)}}$$

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 not follow any preventive action.


IJCTA | JAN-FEB 2012
Available online@www.ijcta.com

ISSN: 2229-6093
Table 1: Defect Density

<table>
<thead>
<tr>
<th>Proj. no.</th>
<th>description</th>
<th>KLOC</th>
<th>No. of defect</th>
<th>Efforts (PHr)</th>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dum Shell CRM module</td>
<td>28</td>
<td>173</td>
<td>2200</td>
<td>0.006</td>
</tr>
<tr>
<td>2</td>
<td>I-lok Automation tool</td>
<td>13</td>
<td>120</td>
<td>1200</td>
<td>0.009</td>
</tr>
<tr>
<td>3</td>
<td>Student attendance Application</td>
<td>6</td>
<td>105</td>
<td>600</td>
<td>0.015</td>
</tr>
</tbody>
</table>

Fig 2

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 2), it is evident that, the number varies.

Table 2. Categorization of defects across phases for three similar projects

<table>
<thead>
<tr>
<th>Life phases</th>
<th>Activity</th>
<th>Defect Type</th>
<th>No. of defect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Review</td>
<td>REQ</td>
<td>47</td>
</tr>
<tr>
<td>Design</td>
<td>Review</td>
<td>DSN</td>
<td>43</td>
</tr>
<tr>
<td>Code</td>
<td>Testing</td>
<td>LOG</td>
<td>255</td>
</tr>
<tr>
<td>GUI</td>
<td>Review</td>
<td>GUI</td>
<td>42</td>
</tr>
<tr>
<td>Documentation</td>
<td>Review</td>
<td>TYP</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 3 Code Description

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description of defect type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ</td>
<td>Requirements</td>
<td>Error in understanding the requirements, or inadequate Requirements definition.</td>
</tr>
<tr>
<td>DSN</td>
<td>Design error</td>
<td>Error in developing design, or inadequate design, or technical inadequacy in design.</td>
</tr>
<tr>
<td>LOG</td>
<td>Logical error</td>
<td>Logical Error</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical error</td>
<td>Error in screen/report layout and design</td>
</tr>
<tr>
<td>TYP</td>
<td>Documentation error</td>
<td>Typographical error in documentation or in code, including spelling errors, mistyped words, and missing delimiters in code.</td>
</tr>
</tbody>
</table>

Table 4: Observed defect pattern across projects

<table>
<thead>
<tr>
<th>Proj. no</th>
<th>logic</th>
<th>Doc.</th>
<th>Req</th>
<th>Design</th>
<th>GUI</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>120</td>
<td>6</td>
<td>20</td>
<td>18</td>
<td>9</td>
<td>173</td>
</tr>
<tr>
<td>P2</td>
<td>76</td>
<td>8</td>
<td>15</td>
<td>11</td>
<td>10</td>
<td>120</td>
</tr>
<tr>
<td>P3</td>
<td>59</td>
<td>7</td>
<td>12</td>
<td>14</td>
<td>13</td>
<td>105</td>
</tr>
</tbody>
</table>
Fig 2: Defect Type pattern across project

Fig 1.2 illustrates the defect type pattern for three 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. 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>
<thead>
<tr>
<th>Sr. no</th>
<th>Defect Type</th>
<th>Root Cause</th>
<th>Preventive Actions</th>
</tr>
</thead>
</table>
| 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.  
• 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.  
• Generally introduction |
| 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 | • 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 |
5) Lack of system knowledge 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 Organization 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

1) 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 three similar projects, and the process improvement was observed in terms of average defect density.

<table>
<thead>
<tr>
<th>Proj. no.</th>
<th>description</th>
<th>KLOC</th>
<th>No. of defect</th>
<th>Efforts (PHr)</th>
<th>Defect Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Employee module</td>
<td>38</td>
<td>119</td>
<td>2900</td>
<td>0.003</td>
</tr>
<tr>
<td>2</td>
<td>Restro reporting</td>
<td>10</td>
<td>55</td>
<td>925</td>
<td>0.005</td>
</tr>
<tr>
<td>3</td>
<td>Call reporting</td>
<td>15</td>
<td>153</td>
<td>1950</td>
<td>0.010</td>
</tr>
</tbody>
</table>

The Graph represents the distribution of defect densities for 3 similar projects before and after implementing the Defect Prevention as provided in the table 1 and table 5 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 4).
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
http://www.chillarege.com/odc/articles/odcconcept/odc.html