ISSN:0975-9646

Defect Prediction and Analysis Using ODC Approach in a Web Application

PRANAYANATH REDDY ANANTULA*,RAGHURAM CHAMARTHI (TEJORAGHURAM) **

** IT-Department, CVR College of Engineering, Ibrahimpatnam, Hyderabad, A.P, India*
***Software Engineer, TATA Consultancy Services, Hyderabad, A.P, India*

***Abstract*-In software project management, there are five basic factors to predict and control, those are size, process, effort, environment and quality. Most of the software engineers focus on these factors to improve the software quality. In practice quality management implies finding defects and rectifying them. Software defects are not well enough understood to provide a clear methodology for avoiding or recovering from them. Most of the research related to software quality focuses on modeling the residual defects in the software to estimate software reliability. Software project management need to be improved in order to predict other possible information about software quality such as in-process defects, their type, classify the defects into a group and so on. Currently software engineers still don't have a complete defect prediction technique for any kind of software product. This paper provides a new approach for predicting the defects and classification of defects based on project characteristics in the early life cycle phases. Using Orthogonal Defect Classification (ODC) ([4], [5]) approach we will see how the current proposed methods are used to improve and realize the quality in development and test environments in a web application.**

Keywords: **Defect Prediction, Retrospection, Orthogonal Defect Classification, Root Cause Analysis**

1. INTRODUCTION

A defect can be determined in software as error, failure, fault, flaw, bug or mistake in a computer program that may deviate or prevent the software from behaving as intended. A defect represents the undesirable aspects of software quality. A complete verification and validation of software is done to ensure that it is fulfilling all the requirements correctly. As exhaustive testing is not possible, the software engineer does several kinds of testing to identify the defects and rectify them before the customer/end-users encounter the defects. It could be an expensive and time consuming process. But we need to find the defects and rectify those defects so that they do not lead to failures after the release of software. Various techniques are already present to do effective verification and validation. But, this research paper presents that this defect prediction technique performs noticeably better in terms of catching the defects in the software at early stages. A defect amplification model can be used to illustrate the generation and detection of defects during the preliminary design, details design and coding stages of software engineering process. During these Phases, defects may be adversely generated, reviews may fail to uncover the newly generated defects and defects from previous phases slide to the next phases, this result in increase of defects. Identifying the errors and avoiding the defects at early stages will result in investing less cost and effort when compare to identifying and removing defects at a later stage. This is explained neatly using Fig 1

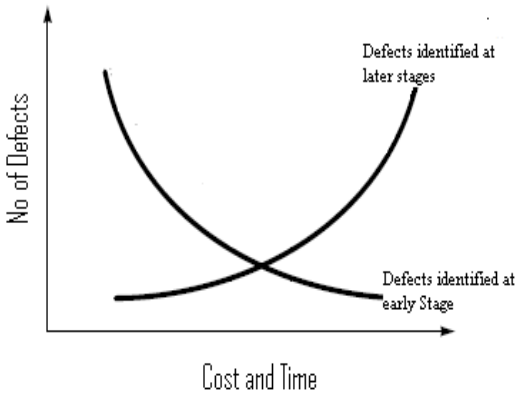

Fig 1 Graph represents Time and Cost required to remove defects

Defect predictors guide the developer through fault prone parts of the software so that the managers can allocate the resources effectively to conduct the testing. Effective use of testing resources at early stage of the process will result in product quality as well as time to market. The most important goal of the defect predictor is to accurately predict the modules where faults are likely to hide in software development life cycle. Accurate predictors could help in reducing time and cost of testing the software.

1.1 Techniques for identifying and analyzing the Defects

Over the years, various types of techniques have been developed to ensure the quality of software such as.

- Defect Detection Technique
- Defect Analysis Technique
- Defect Prediction Technique

Defect Detection Technique

Variety of testing techniques can be used to identify software defects in software development process for all domains, functional and non-functional requirements (such as boundary testing, component testing, scalability,

interoperability, stress testing, code inspection, alpha testing, beta testing).Defect detection technique such as these are used in most of the software development projects to discover the defects, document those defects for improving quality of the product,. The defect document information will be an input for defect analysis technique. Defect density in the software can be identified using

Defect Density (DD) = Sum of the defects found during all the test cycle / KSLOC

Note: The defect density is not always proportional to the size and complexity.

Defect Analysis Technique

Analyzing the defect leads to identifying the root cause of the defect and to find the solution to overcome the defect in further development process. This analysis will be useful to improve the quality of the software and even the productivity of the developer.

Some of the defect analysis techniques are

- Defect Classification using Defect taxonomies
- Root Cause Analysis
- Fish Bone Analysis

Defect Prediction Technique

The goal of defect prediction technique is to anticipate and prevent defects proactively before they can occur and cause failures. This can be done by experience of the predictor in the same project or in the relevant kind of projects. By using the experience gained from the earlier projects software engineers can identify the common causes for defects in work products and change the process to eliminate those causes to be occurred. The resulting form of successful application of defect prediction technique can be reapplied elsewhere in the project life cycle.

Some of the defect prediction models are:

COQUALMO

Constructive quality model predicts the defect density of the software under development where defects conceptually flow into the defect removal pipes so that the defect does not flow from one phase to another phase and become a big defect that may lead to project chaos.

Mining Test Defects using ODC

Defect analysis technique like Root Cause Analysis, Statistical Growth Modeling (e.g. s-curves) [3] and Fish Bone Technique play a useful role in analyzing the software defects and in mining the defects.

ODC was invented by IBM research to establish a Foundation for providing analysis and feedback of defect data targeting quality issues in software design and code in a procedural language environment.

Why defect prediction is important?

1) Avoiding defects at an early stage eliminates wastage in process and development of product

Defects are the leading contributors to IT waste and cause significant project rework, delay and cost over runs. In the current trend the projects are speeding to meet time to market in this process. Majority of defects are not identified until the testing phase of a project. In many cases defects are escaped into production that lead to more rework and also it increases cost for rework. And when analyzed they found that most of the defects originated in the requirement phase of the project. If these defects are identified, less rework will be done at later stage of the software development process. The errors at requirement phase occur due to poorly defining the requirements and due to ambiguous requirements.

2) Cost Efficiency

Most of the software product seems to contain a lot of bugs, particularly in early release because engineering team's are often forced to work towards the release deadlines that are not reasonably set. To meet the deadlines, team members cut corner, skip steps or fail to look hard for mistakes and release the product with known bugs. Once the product has been released errors continue to add cost to the product. The number of defects identified by the customer is communicated to the product support team. More errors are found more rework is required and it leads to customer dissatisfaction. The more complex the problems, the more expensive the support personals required. The errors found by the customer have an added effect on the revenue in addition to the increased cost. Customer will likely not volunteer to give references or aid more sales and customer will be likely to replace the buggy product sooner than later. One obvious solution is to never set unreasonable release dates, establish effective engineering process and eliminate errors as early as possible in software development life cycle.

With the current intent in the process improvement many organizations have positioned themselves to begin to improve their testing practices. This is evident in many of the organizational budgets for continuing process improvement, acquiring testing tools and trainings their resources.

2. RELATED WORK

ODC is a measurable system for software process based on semantic information contained in the defect stream. It brings a new level of sophistication for analysis and leverage information captured in the development process. ODC technology works by leveraging information contained in software bugs-something quite freely and often abundantly available in any software development process. However, the entry point of ODC is not rigidly tied to organizational maturity. ODC helps in achieving and maintaining high maturity with less expense and greater control [6].

ODC is a scheme to capture the semantics of each software defect quickly. It is the definition and capture of defects attribute that make mathematical analysis and modeling possible. Analysis of ODC data provides a valuable diagnostics and classification method for evaluating the various phases of the software life cycle and the maturity of the product [7].

The following case study uses ODC classification mechanism to analyze the efficiency of the software testing process.

Classifying the process based on the kind of defect that triggered. Triggers are a key to give an understanding of what happened during a test cycle and its efficiency. Additional data such as ODC defect types and ODC source would be useful to further narrow down to the specific development practice problems. ODC trigger can be easily generated from the defect tracking logs. These logs are created while analyzing the defects.

A defect analysis conducts a postmortem, also called as retrospection to identify the root cause of the defect. Retrospective analysis generates trigger that tends to be easy to extract from test defect logs. The Classification is done based on Context, User Interface, Components, Navigation, Configuration, Security, Performance and Process. Each classification in turn triggers some activities for testing.

2.1 Applying ODC to In-Visualizer Web Application.

The project is large sized, contains of more than 1000KLOC. The project employs the incremental process model. In ODC activity, the tester will be inputting the data and developers will be responsible for the end actions.

Steps involved in adopting ODC in test cycle are

Step1: Initially defect data is collected from all possible phases of testing.

Step2: Analysis is needed on each defect data for better understanding.

Step3: Adding ODC attribute names for activities, triggers and impact of defect on data

Step4: classify ODC scheme differentiates and then group defect data which are independent or no redundant.

Step5: Deriving information from the classified defects by applying domain experience based.

These steps lead as a road map for improving and doing the testing.

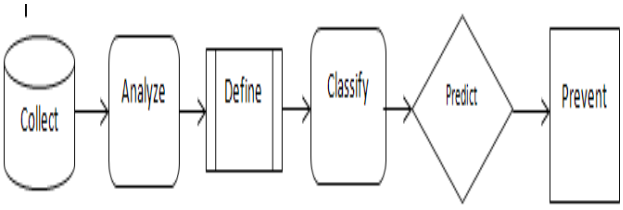

Fig 2 Steps to implement ODC

The history of the defect data will be collected from the defect tracking tools and ODC tags will be inserted. ODC tags will be used in the subsequent data input steps.

Before starting with the test execution cycle in ODC, ODC team should customize the tags in accordance with the In-Visualizer web application. Table 1 shows the customized ODC tags.

Activities	Triggers	Impact
Requirement Review and Inspection	Completeness Flow Consistency Sequencing Coverage Behavior Conform Clarity	Correctness Concerns Completeness Concerns Usability Concerns
Design & Code Review	Design Flow Backward-Compatibility Concurrency Interaction Package Sequencing Variation	Correctness Concerns Reliability Concerns Data Integrity Concerns Consistency Concerns
Integration Testing	Flow Interaction Navigation Link Complex	Correctness Concerns Reliability Concerns Data integrity Concerns Recoverability Concerns
System Testing	Navigation Behavior Accuracy Workload Recover Restart Interaction Backward-Compatibility	Security Concerns Performance Concerns Correctness Concerns Reliability concerns Recoverability Concerns
Usability Testing	Behavior Aesthetics Style Input Language	Usability Concerns Correctness Concerns

Table 1 ODC Tags for In-Visualizer Web application

3. RESULTS AND ANALYSIS

Several test strategies are iterated at regular intervals. During every interval, the testers feed the ODC tags into the defect tracking management tool. Various defects distribution charts are prepared based on the defect tracking tools and ODC tags. The charts are prepared by using defect distribution data after several test execution cycles. Through the defect tracking tool the authors queried the defects data across all the test cycles, without including the ODC tags and make the charts Fig 3.

Defect distribution indicates percentage of defects found during each phase of SDLC. Having looked at the defect distribution chart, it is easy to know the phases where defect distribution is high.

Now to improve the quality they have paid more attention in removing the defects in requirements and design. The evaluation of defect areas can also be done based on severity of defects Fig 4. Charts can be used to get a look at the number of defects by category, such as priority or severity.

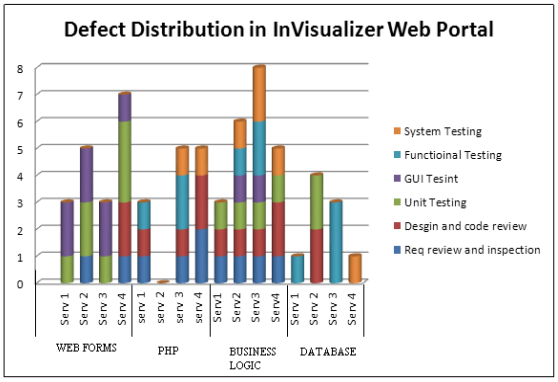

Fig 3. Defects Distribution across various Testing types and its severity

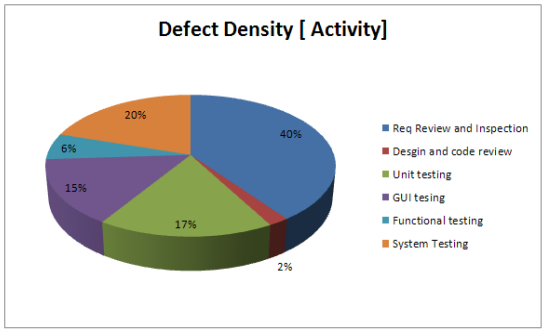

Fig 4. Defect Density across various ODC activites and severity

The inference made or lesson learnt even after the exhaustive analysis on the defect data is inadequate to derive information to disseminate among all members.

This situation induces to choose and change the objective of mining the defect data to add more sense. Hence they queried the defects data across all the test cycles and generated the charts including the ODC tags.

The authors in this paper have taken 3 major web components of In-Visualizer web application which contributes to major area of defects. Fig 5 depicts the following points.

1) Most of the defects are uncovered during ODC activity- System Testing
2) System testing activity shares the major part in web forms component out of all 3 chosen components.

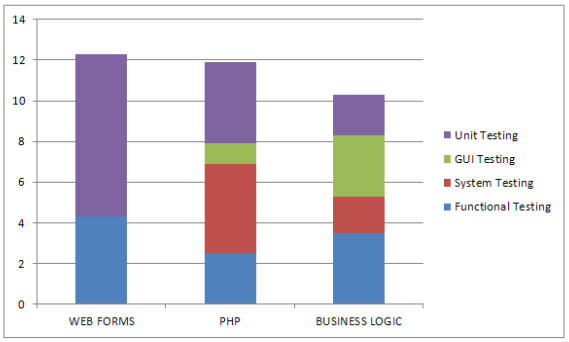

Fig 5. Defect Distribution across Modules

Combination of these inferences and pareto principle help the management to reduce the major portion of the defects by acting on the right areas

4. CONCLUSION

Software defect prediction work focuses on the number of defects remaining in a software system. This paper provides a detail report of a software process model which will help in predicting the defects by classification and evaluation. It also provides a detail description of how the software process model aligns itself to business goals and achieves good quality software by predicting the number and type of defects well in advance and take necessary action to reduce the occurrence of defects. ODC results help to predict the software defects and assist project managers in allocating testing resources effectively to improve the quality of the software.

REFERENCES

[1]. Butcher (2002), "*Improving software testing via ODC: Three Case Studies*", M.Butcher, H. Munro, T.Kratschmer, IBM Systems Journal, Vol.41, No.1

[2]. Brad (2001)," *How good is the software: A Review of Defect prediction Techniques*", Brad Clark, Dave Zubrow, Carrnegie Mellon University

[3] Mullen (2002) "Orthogonal *Defect Classification at CISCO*", T.Mullen, D.Hsiao, Proceedings ASM conference.

[4] Ram, "*Orthogonal Defect Classification*". www.Chillaregte.com/odc

[5] Ram (1992), "*Adapting ODC to improve software quality: A case Study*", Yang Gu, Software Engineer, IBM http://www.ibm.com

[6] Yang (1992), "*Orthogonal Defect Classification A Concept for In-Process Measurements*", Ram Chillarege, IEEE Transactions on software Engineering, Vol 18, No.11, November.

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

[8] Chillarege (2002) ,"*Test and development process retrospective 0a case study using ODC triggers*" , Chillarege, R.; Ram Prasad, K.'