DEFECT PREVENTION BASED ON 5 DIMENSIONS OF DEFECT ORIGIN

Sakthi Kumaresh[1] and Baskaran Ramachandran[2]

[1]Department of Computer Science, MOP Vaishnav College, Chennai, India
sakthimegha@yahoo.co.in
[2]Baskaran Ramachandran, Anna University, Chennai, India

ABSTRACT

"Discovering the unexpected is more important than confirming the known [7]. In software development, the "unexpected" one relates to defects. These defects when unattended would cause failure to the product and risk to the users. The increasing dependency of society on software and the crucial consequences that a failure can cause requires the need to find out the defects at the origin itself. Based on the lessons learnt from the earlier set of projects, a defect framework highlighting the 5 Dimensions (Ds) of defect origin is proposed in this work. The defect framework is based on analyzing the defects that had emerged from various stages of software development like Requirements, Design, Coding, Testing and Timeline (defects due to lack of time during development). This study is not limited to just identifying the origin of defects at various phases of software development but also finds out the reasons for such defects, and defect preventive (DP) measures are proposed for each type of defect. This work can help practitioners choose effective defect avoidance measures.

In addition to arriving at defect framework, this work also proposes a defect injection metric based on severity of the defect rather than just defect count, which gives the number of adjusted defects produced by a project at various phases. The defect injection metric value, once calculated, serves as a yardstick to make a comparison in the improvements made in the software process development between similar set of projects.

KEYWORDS

Defect, Defect avoidance, Defect Injection Metric, Defect Severity

1. INTRODUCTION

Profound confusion prevails at the final stages of software development as to which defect discovered belongs to which phase of software development life cycle. Techniques like root-cause analysis and orthogonal defect classification are some of the commonly used practices. These Techniques are applied jointly with the software process to determine each defect attribute in terms of its type, trigger, source etc [3]. However, the need is to find a mechanism to determine the origin of those defects and ways to eliminate them at origin itself and the same has been realised in this study by way of proposing defect prevention mechanism for each defect type under 5Ds.

Everybody likes Defect prevention as nobody likes defects, especially those defects that are found by our customers. We need to stop defects from reaching our customers by catching them before

DOI : 10.5121/ijsea.2012.3407

we deliver them to our customer [2]. Nevertheless, having one vague value in terms of percentage of defects found for each phase have proved to be an inaccurate way due to the variations of defect severities and impact to the software schedule, effort and quality [3].

Due to this, a severity factor for various defects with respect to their impact to the software schedule, effort and quality has been taken into consideration for calculation of defect injection metric. This paper proposes defect injection metric that finds out the defect count under each phase of software development and for the project as a whole. This work also suggests ways to reduce the defect count on subsequent projects. The defect injection metric value, once calculated, will serve as a common denominator for comparison between projects as well as one of the best parameters as a Quality Metric.

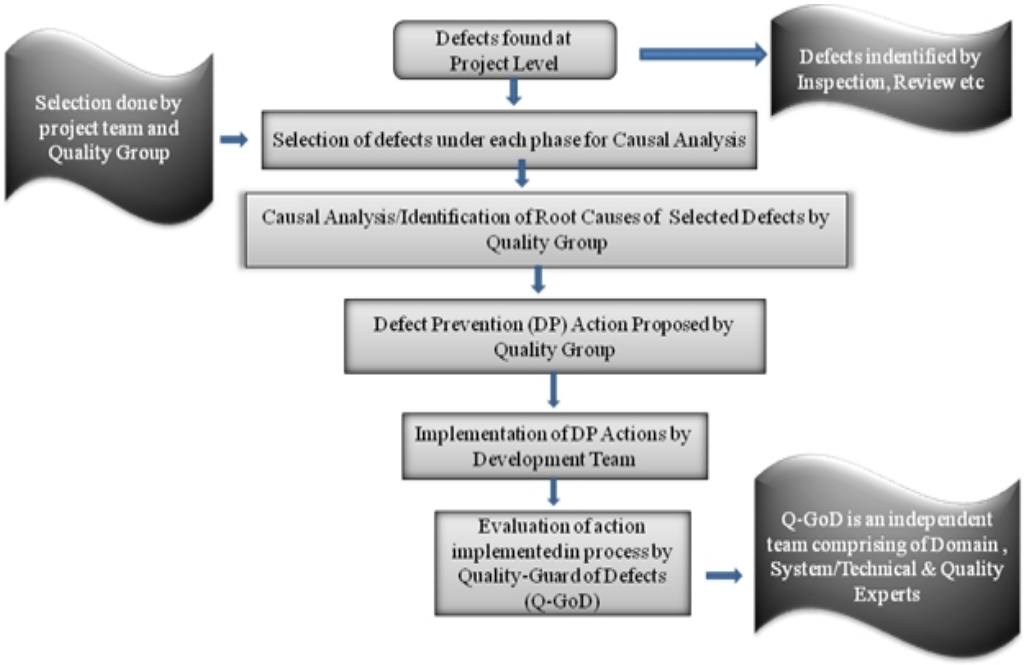

Figure 1 shows the work flow involved in this study for defect prevention.

2. SOFTWARE DEFECT FRAMEWORK

The software defect framework highlighting the 5 Ds of defect origin is proposed in this work. Each one of the D's concentrates on defects in one particular stage of software development lifecycle like Requirements, Design, Coding, Testing and defects due to timeline problem. After doing a thorough analysis of various defect types under each stage of software development, the most prominent defects are identified. The defects are then prioritized based on their importance and the top six defects from each category are taken for study. Certain type of these defects are marked with Explicit (E) to signify the in-process influence and certain other defect types are marked with Implicit (I) to signify those defects that are reported by customer. Utmost care should be given to such defect types in order to satisfy customers. For each one of the identified defect type, the reason for such defect is found out and the DP actions are suggested. These DP actions, when introduced at all stages of a software lifecycle, can reduce the time and resource necessary to develop high quality systems. Figure 2 depicts the software defect framework.

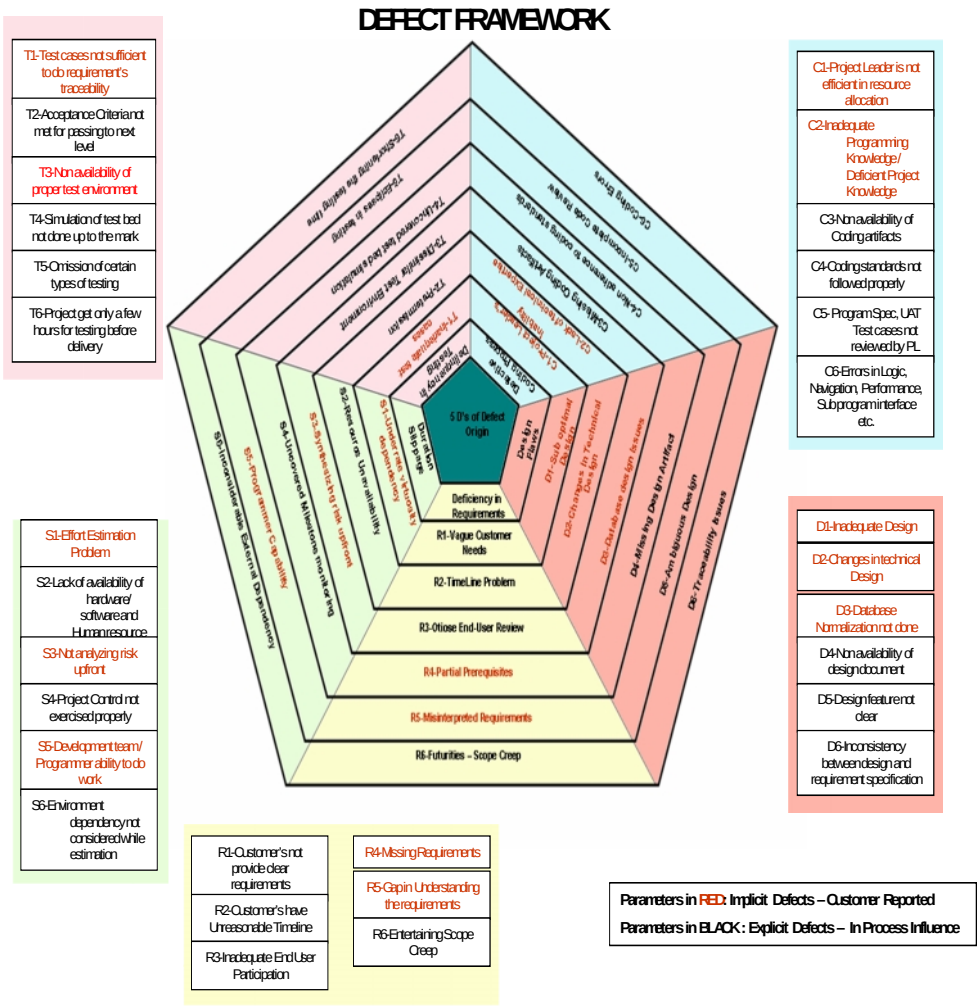

Figure 2: Software Defect Framework

2.1 Deficiency in Requirements (D1)

We accept that testing the software is an integral part of building a system. However, if the software is based on inaccurate requirements then, despite well-written code, the software will be unsatisfactory. Instead of limiting our testing to code, we should start testing as soon as we start work on the requirements for a product [5]. The aim of this work is to find requirements-related defects as early as they can be identified and hence prevent them from being incorporated in the design and implementation. The list of top six defects identified at the requirement stage with its root causes and defect prevention action to be taken [8] for such type of defects is shown in the Table 1.

Table 1: Requirement Defects with their root causes and Defect Prevention action

Pain Points	Root Causes	Defect Prevention
Vague Customer Needs	Business Analyst (BA) not well Qualified	Extensive Training to BA in the domain in which he/she is working should be given
TimeLine Problem	Accepting unreasonable timeline of the customer	Customer timeline should be accepted after doing an In depth feasibility study of the project
Inadequate End User Review	End User not spending quality time for review	The requirements should be made to sign off properly by the end user
Partial Prerequisites	Not using a formal requirements checklist for analysis	Quality Team should validate if requirements checklist is followed by BA
Misinterpreted Requirements	Communication gap between project stakeholders	The BA and the PM should play a vital role to make the Development Team understand the Customer domain and the project requirements..
Futurities	Add-on to the existing requirement, without knowing it is a new requirement	The BA along with the PM should freeze the scope of the project before starting the development.

2.2 Design Flaws (D2)

The design should be perfect for software development. It is observed that many projects fail due to poor design. It may be architectural design, conceptual design, database design etc. Much software has been developed, installed and passed to maintenance with a faulty design. The faulty design should be detected and corrected in order to have high customer satisfaction [5]. List of defects that would emerge out of design phase with their root causes and steps to be taken to avoid such defects are given in the form of preventive actions and the same is listed in Table 2[8].

Table 2: Design Defects with their root causes and Defect Prevention action

Pain Points	Root Causes	Defect Prevention
Sub Optimal Design	Design features described does not provide the best approach (optimal approach) towards the solution required	PM and Design experts in the Organization should formalize the best design approach before starting the development of the project.
Change in Technical Design	The design change would have a maximum impact when new requirement is introduced at a later stage of the project	Any new requirement which has impact on the design should undergo formal Configuration management process to take up the new requirement in the next phase of the project.
Database Design Issues	The design team might not have the expertise to handle the normalization.	PM can seek the help of Design experts in the Organization to handle normalization.
Missing Design Artefact	The Design template may have many sections and only the major mandatory sections might have got covered.	The Quality Team should find out the reason as why only certain sections are

		considered as mandatory and the reason should be justifiable
Ambiguous Design	Design document includes ambiguous use of words or unclear design features	PM can seek help of Document writers to help the Design Team to write the unambiguous documents
Traceability Issues	Inconsistencies between design and requirement specification	PM should start mapping the Traceability matrix on completion of the Design document, so that these issues can be handled upfront.

2.3 Defective Coding Process (D3)

It is observed that many applications reach the final stages without unit testing; review of unit test cases, etc. Due to the lack of these processes, the application is released with uncovered defects, which could have been rectified at an early stage if proper coding processes were in place. If such an application is given to the customer, it will be returned for proper recertification and revalidation [5]. This proves to be costly for the customer as well as the organization. Top six defects identified at coding stage are shown in Table 3[8].

Table 3: Coding Defects with their root causes and Defect Prevention action

Pain Points	Root Causes	Defect Prevention
Project Leader's Inability in Resource Allocation	Inefficient Work Break Down (WBS) structure	PM should review the WBS assigned by the PL and should take mitigation steps if he foresees any risk in it.
Lack of Technical Expertise	Project team members are not trained properly (or) may not have worked in similar technical domain.	Skilled resources with Technical work experience should be inducted into the project, if project has tight deadlines.
Missing Coding Artefact	Due to shortage of time	PL should find reasons for time constraints and should talk with PM to see if the estimated effort has to be relooked for sufficiency
Non adherence to coding standards	To complete the code fast and due to time constraints, developers may not adhere to coding standards	PM/Quality Team can cross check if PL is doing the code review or not.
Incomplete code review	PL might feel that testing the code would suffice and may skip the review.	Quality Team can do audits / checks to see if review is done formally.
Coding Errors	Programmers not familiar in developing similar type of codes or errors may be due to careless mistakes	Training should have been provided before the development starts.

2.4 Delinquency in Testing (D4)

The end of phase reviews or inspection is the key to removal of defects for the development phases; however, for testing phases the testing by itself is the key [9]. When the defects identified by testing are not dealt correctly, it may cause new defect injections. Table 4 lists top six defects that would emerge out of testing phase [8].

Table 4: Testing Defects with their root causes and Defect Prevention action

Pain Points	Root Causes	Defect Prevention
Inadequate test cases	The testers may lack knowledge on the possible scenarios of test path for the requirements.	The BA should run-through all possible scenarios of the actual Business to the Test team so that the Testers can build a strong Test Case base for the project
Pretermission	Due to lack of time and schedule slippage, pressure might mount on the team to move to the next level of testing.	The PM shouldn't hide the facts as up to what level the Testing is performed and should involve the client for a joint decision.
Dissimilar test Environment	Space restrictions in the Test machine	Space restrictions, if any, should be handled by PM and client mutually.
Uncovered test bed simulation	Unavailability of Database space may restrict the Test bed setup.	Size of database required for testing has to be planned (by PM & PL) upfront and sufficient time should be given to the IT Team for DB space allocation.
Eclipses in testing	Due to unavailability of the Testing tool S/W, Experienced Testers not available.	While planning the project, the PM should decide (along with the client) how these tests can be performed Testers with expertise in such type of testing can be used from other teams in the Organization
Shortening the testing time	Inadequate testing	Proper Planning should be done before start of SDLC activities.

2.5 Duration Slippage (D5)

Software is developed to automate a set of business processes, but the requirements change so frequently that the project gets far behind schedule and the output of the system becomes unreliable. Periodically, the developer is pulled off the project in order to incorporate all the requirements, which makes them fall even further behind the schedule. Due to this busy schedule, there may be many defects in the application [5]. It may not be possible to capture all the defects, but the most prominent one are listed in Table 5.

Table 5: Reasons for Duration Slippage

Pain Points	Root Causes	Defect Prevention
Synthesising risk upfront	PM may not be analysing all possible risks.	PM should analyse all the risk factors related to the project like People, Environment, Scope, Client, Sub vendors, etc and all subcategories to it.
Programmer Capability	Programmer may not be Technically capable to finish the work in the allotted time.	Skilled programmers are to be employed for tasks in the critical path.
Uncovered Milestone monitoring	Project control not exercised properly / Monitoring milestones not done.	PM has to track in the Project Management Tool and Quality / Senior Management Team should do periodic reviews on this.
Entertaining scope creep	PM may not distinguish if it is a new requirement (or) an addendum to the existing requirement.	PM should have the overall control of the project
Resource Unavailability	PM not judging on the exact resources requirement for the project.	PM should plan for Human and non-human resources in the initial planning stage of the project itself.
Inconsiderable External Dependency	PM / PL may under estimate for external dependencies.	PM / PL should always have some buffer while planning for external dependencies.

3 DEFECT INJECTION METRIC

Software measurement is concerned with deriving a numeric value for some attribute of a software product or a software process. Comparing these values to each other and to relevant standards allows drawing conclusions about the quality of software product or the effectiveness of software processes [4]. Working on this line, in order to know the effectiveness of software processes, Defect Injection Metric based on defect severity is proposed in this study.

3.1 Defect Severity

The severity is the extent to which a defect causes a failure that degrades the expected operational capabilities of the system of which the software is a component. The severity is classified from the point of view of effect on business efficiency. Following classification of defect severity assumes that the software remains operational [6].

Table 6: Classification of defect severity

High(h)	Weightage factor - 1	These are the extremely severe defects, which have already halted or capable of halting the operation of business system
Medium(m)	Weightage factor - 3	These are also severe defects, which have not halted the system, but have seriously degraded the performance of some business operation
Low(l)	Weightage factor - 5	These types of defects are the ones which are primarily related to the presentation or the layout of the data. However, there is no danger of corruption of data and incorrect values

3.2 Process of Obtaining Defect Count

In order to obtain the defect count at every phase of software development, the defect type under each phase is considered. Later, these defect types are categorised into Low, Medium and High according to the severity of the defect type. Weights are assigned to this scale of value as shown in the table 6.

3.2.1 Calculation of Defect Injection Metric

The severity of the defects based on the impact of the defect in terms of timeline, risks, debugging etc., makes some defects either truly negligible or to have a major impact. To adjust the effects of such defects, a severity factor has been introduced to adjust the defect count to be more realistic for our metrics purposes. Defect count for each defect type under each stage is calculated using the formula 1

Defect Injection Metric (Stage X) = $\sum_{j=1}^{6} DC$ ---------- (1)

Where DC = $1\,X_l + 3\,X_m + 5\,X_h$ and X = Number of defects

Defect Injection Metric would then be arrived for entire project as follows:

Defect Injection Metric (Project) TD$C = \sum_{I=i}^{6} \sum_{j=1}^{5} DC$ ---------- (2)

Where TDC = Total Defect count

4 PROJECT CASE STUDY:

This study has been performed for telecom applications developed in Java in the year 2010. The size of each application was around 100 KLOC. Defect detection activities like Inspection, Review, testing etc were carried out to find out defects. Later these defects were analyzed for defect counts, their severity, and the same has been listed. Based on the Defect prevention action suggested in this study, enhancement was done and the defect count was checked for a similar size project in the year 2011 and the findings are tabulated in the Table 7& Table 8. Table 7 is the calculation for DC for requirements stage (D1) alone and the results of similar calculations for D2-D5 are shown in Table 8.

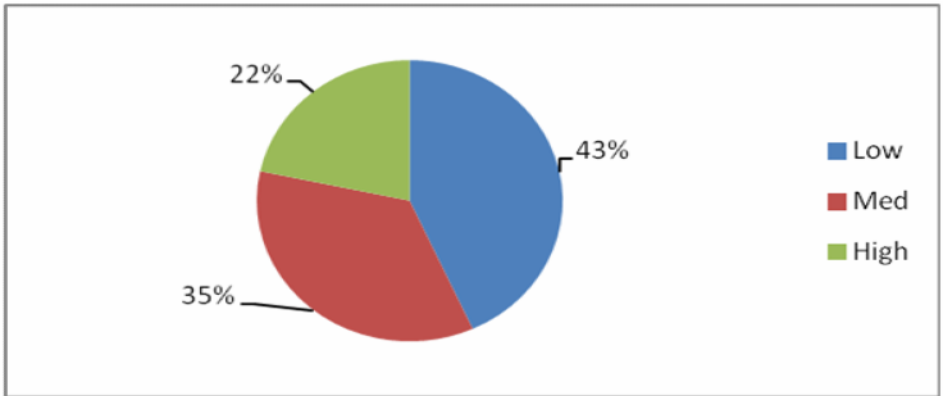

Figure 3: Defect Distribution per severity for D1

Table 7: Comparison of Defect Count values for D1

	2010 Project	*Before Defect Prevention*						*2011 Project -After Defect Prevention*				
S No	Defect Type	*No of Defect*	Defect Severity			Defect Count		*No of Defect*	Defect Severity			Defect Count
			L	M	H				L	M	H	
D1	***Deficiency in Requirements***											
1	Vague Customer Needs	6	2	3	1	16		4	1	2	1	12
2	TimeLine Problem	5	2	2	1	13		4	2	1	1	10
3	Inadequate End User Review	4	2	1	1	10		2	1	1	0	4
4	Partial Prerequisites	8	3	4	1	20		6	2	3	1	16
5	Misinterpreted Requirements	8	4	2	2	20		5	2	1	2	15
6	Futurities	6	3	1	2	16		6	3	2	1	14
					∑DC	**95**					***∑DC***	**71**

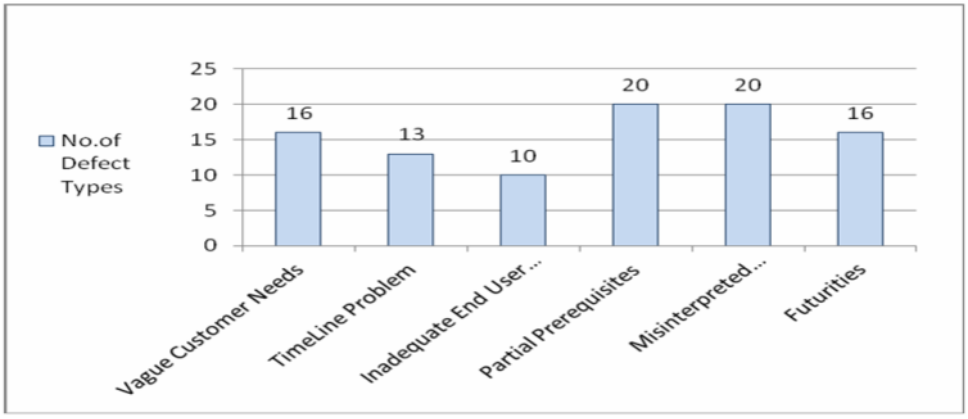

Figure 4: Defect Distribution across various defect types for D1

Table 8: Comparison of Defect Count values for D1-D5

	2010 Project ---- Before DP Action			2011 Project -- After DP Action	
S No	5 Ds	No of Defects	∑DC	No of Defects	∑DC
D1	Deficiency in Requirements	38	95	28	71
D2	Design Flaws	49	115	32	93
D3	Defective Coding	62	165	39	98
D4	Delinquency in Testing	45	126	34	73
D5	Duration Slippage	45	116	31	71
		∑TDC	617	∑TDC 406	

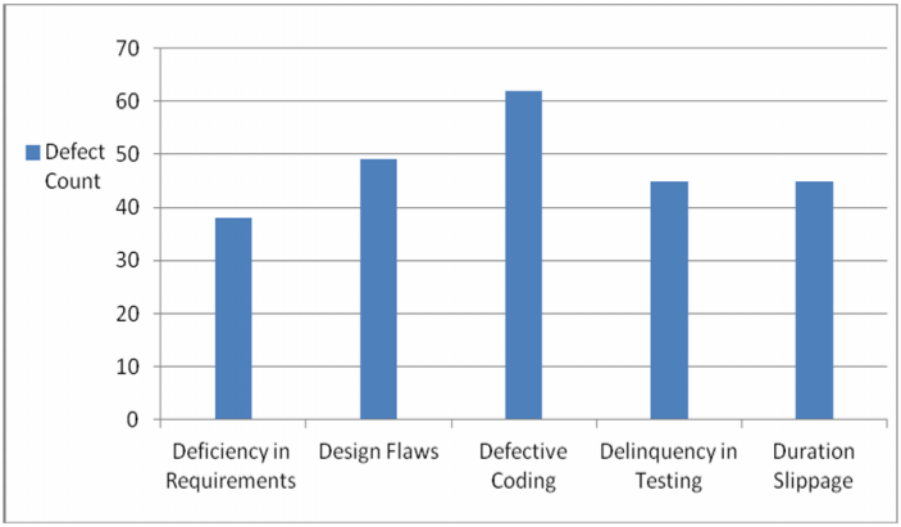

Figure 5: Defect Distribution across 5D before DP action

4.1 Overall Results

Based on defect type, the appropriate defect prevention action as suggested in this study was implemented in the set of projects done during the year 2011. Upon application of defect injection metric on 5Ds, the defect counts were found out and compared with the earlier set of projects. The considerable reduction in the defect count values for 2011 project is evident from the table 9 and the same is depicted in the figure 6

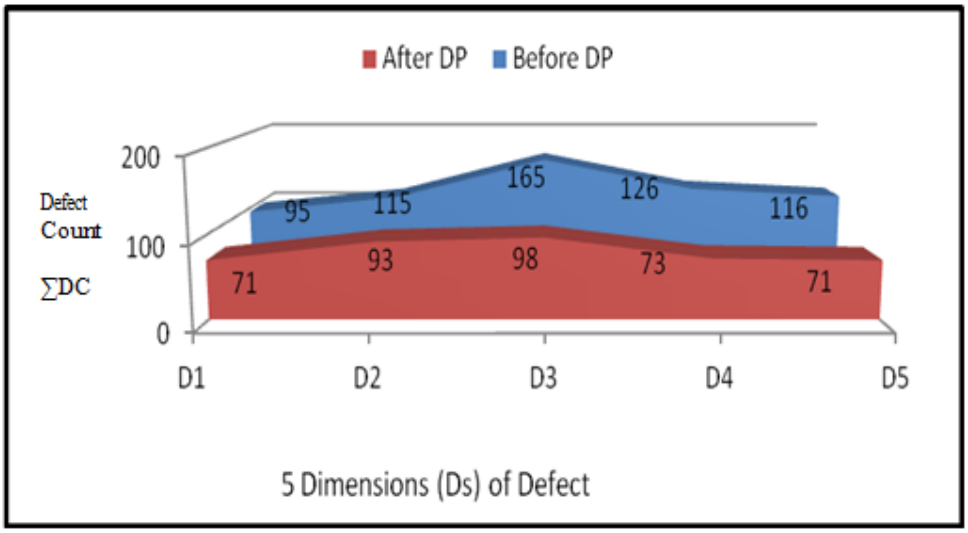

Figure 6: Comparison of Defect Count before and after Defect Prevention Actions

5 CONCLUSION

This research work provides a general framework of defect with its defect prevention measures suggested in order to enhance quality culture establishment in an organization. Implementation of defect prevention measures in subsequent projects would result in better performance, rapid and sustained improvement in the product quality as is evident from the example. To get such results, organization should fully consider the product Characteristics, make defect prevention activities responsibilities for each stage of software development and demonstrate a firm senior management commitment by employing a special independent team like Q-GoD (Quality- Guard of Defects) to enforce strict quality traits under each phase of software development[8].

The Defect preventive actions that are proposed in this paper are limited to only few types of defects under each category. There may be many other defects that would evolve at each stage; but these defects are found to be more prominent as per the practitioner's experience and hence they are concentrated in this study. This study could be extended by adopting ODC way of classifying defects [1], so that minute details about the defect can be captured and better defect prevention action can be arrived out. When these DP actions are implemented in the project, the quality of the project can be further enhanced.

REFERENCES

[1] Sakthi Kumaresh, R Baskaran (2010) "Defect Analysis and Prevention for Software Process Quality Improvement", International Journal of Computer Applications(0975-8887) volume 8-No 7, pages 42-47

[2] Omar Alshathry, Helge Janicke, Hussein Zedan, Hussein" (2009), "Quantitative Quality Assurance Approach" Proceedings of IEEE International Conference on New Trends in Information and Service Science, Page 405-408.

[3] Meng Li, He Xiaoyuan, Ashok Sontakke (2006) "Defect Prevention: A General Framework and its Applications" Proceedings of the IEEE sixth International Conference on Quality Software (QSIC06) .

[4] http://www.scribd.com/api_user_11797_Master%20Of%20Earth/d/7010681-Software-Quality-Metrics
[5] 7 "S" of Defects Occurrence – A Case Study, Arupratan Santra
[6] Software Testing Genius – In & Out of software testing under one roof http://www.softwaretestinggenius.com/articalDetails?qry=716
[7] George Box, Stuart Hunter, "25 Great Quotes for Software Testers" http://hexawise.wordpress.com/2010/01/26/25-great-quotes-for-software-testers/
[8] Sakthi Kumaresh, Baskaran Ramachandran (2012) " Experimental Design on Defect Analysis for Software Process Quality Improvement", Proceedings of IEEE International Conference on Recent Advances in Computing and Software Systems" (RACSS)" Pages 293-298.
[9] Stephen H Kan, Metrics and Models in Software Quality Engineering, Pearson Education, 2nd Edition, 2003

Authors

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 specialisation include Software Engineering, Software Project Management, Software Testing, Software Quality Management and Unified Modelling Language. She is doing research in the area of Software Quality Engineering. She has publications in National conferences and International journal.

Dr. Baskaran Ramachandran is working as the Assistant 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