Study on Software Quality improvement based on Root Cause & Corrective Actions

Abhishek Anurag
Ph.D. Student in Computer Science & Engineering
Amity School of Engineering & Technology
Amity University Mumbai

1. Abstract:

Complexity of software system has grown exponentially as usage and nature of software has changed significantly. In such complex and ever-changing environment, it’s very difficult to maintain the Quality of the software system. Designing, developing, maintaining a software system and ensuring its quality is very costly as cost is directly associated with efforts & time.

Whenever a software system is used by users and if quality of the system is poor they will be encountering issues (delivered defects) which needs to be fixed. Such issues impacting quality must always be “Root Caused” to see why they originated, how they didn’t appear while designing, developing or testing the system. It’s very important that issues are caught very early as late an issue is caught it’s very expensive to fix that. There might be many reasons of issues to be missed viz. unit level testing was not done correctly, code coverage was not proper, test planning was not good, test execution didn’t happen, etc. In a very simple terminology it can be said that “There must always be a test case which can replicate the issue found in the system before an end user reports that”. Once issue is ‘identified and root caused’ then exact “Corrective Actions” can be taken. It will help ensure in future for the same or similar systems having similar design principles are not yielding similar failures.

2. Keywords:

Defect Detection, Defect Prevention, Root Cause, Corrective Action, Life Cycle

3. Introduction:

3.1 Overview:

Quality is directly associated with ‘Function Points’ which was defined in 1979 in ‘Measuring Application Development Productivity’ by Allan Albrecht [3] at IBM. Function point is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user. Function points measure software size. Software size is directly associated with complexity and quality.

As per studies on “Software Quality in 2012: A Survey of the State of the Art” by Capers Jones [4], Vice President and Chief Technology Officer, Namcook Analytics LLC, total defects delivered per function point for best in class, average and poor-quality software systems are as below:

	AVERAGES FOR SOFTWARE QUALITY		
Defect Origins	Defect Potential	Removal Efficiency	Delivered Defects
Requirements	1	77%	0.23
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Documents	0.6	80%	0.12
Bad Fixes	0.4	70%	0.12
TOTAL	5	85%	0.75
Defects per Function Point		**Pass % = 85%**	**Fail % = 15%**
	BEST IN CLASS SOFTWARE QUALITY		
Defect Origins	Defect Potential	Removal Efficiency	Delivered Defects
Requirements	0.4	85%	0.06
Design	0.6	97%	0.02
Coding	1	99%	0.01
Documents	0.4	98%	0.01
Bad Fixes	0.1	95%	0.01
TOTAL	2.5	96%	0.10
Defects per Function Point		**Pass % = 96%**	**Fail % = 4%**
	POOR SOFTWARE QUALITY – MALPRACTICE		
Defect Origins	Defect Potential	Removal Efficiency	Delivered Defects
Requirements	1.5	50%	0.75
Design	2.2	50%	1.10
Coding	2.5	80%	0.5
Documents	1	70%	0.30
Bad Fixes	0.8	50%	0.40
TOTAL	8	62%	3.04
Defects per Function Point		**Pass % = 62%**	**Fail % = 38%**

It's very difficult and complex to define quality of a software system. There are numerous ways it's defined and achieved in software systems. A software system can have different level of tests. Every test plans will have finite set of test cases and upon execution will give Q-Score. This tells overall quality of the system based on different areas tested depending upon test plan's Q-Score. It's really very complex and costly to ensure quality of system is not degrading based on new features implementation, feature modifications and bug fixes due to changing requirements and environments. Such changes must not cause new bugs or regressions. Any bug in the system cause the system quality to degrade as end user's exquisite experience goes down with every bug encountered in the system. So, for engineers involved in development and validation of software system it's very critical to ensure Q-Score is always as desired and acceptable. Every system can have different level of Q-Score criteria based on the milestones of the product viz. Alpha, Beta, RC, PRC, MP, OTA, etc.

Quality of a software system can typically be measured as Q-Score. Q-Score or Quality Score or Pass % can be defined as:
Q-Score == [Total Tests Passed / Total Tests Executed]. It's a metric which eventually tells the health of the software system in its current state.

Traditionally a Q-Score matrix can be defined as below:

Test Plan	Alpha Q Score Criteria	Beta Q Score Criteria	RC Q Score Criteria	PRC Q Score Criteria	MP Q Score Criteria	OTA Q Score Criteria
Smoke Test Plan	70%	80%	90%	100%	100%	100%
Sanity Test Plan	60%	70%	80%	90%	100%	100%
Product Test Plan	NA	60%	70%	80%	90%	95%
Performance Test Plan	NA	60%	70%	80%	90%	95%
Power Test Plan	NA	60%	70%	80%	90%	95%
Reliability (Stress & Stability) Test Plan	NA	60%	70%	80%	90%	95%
Day of Usage (Battery rundown) Test Plan	NA	NA	60%	70%	80%	90%
Interoperability Test Plan (IOT)	NA	NA	60%	70%	80%	90%
Ecosystem Test Plan (Apps/Games/Sites/Contents)	NA	NA	60%	70%	80%	90%
Use Experience Matrix (Subjective & Objective)	NA	NA	60%	70%	80%	90%
Google Certs (CTS, CTS Verifier, GTS)	NA	NA	90%	100%	100%	100%

Above matrix example can be any traditionally ideal Q-Score metric for Android based Embedded System Software running on embedded platforms. NA is "Not Applicable" test plans for that milestone. Milestones can be briefly explained as: 'Alpha' is first phase of software development or bring up stage, 'Beta' is features complete stage, 'RC' is release candidate for internal beta users, 'PRC' is production release quality which will go to initial device manufacturing stage, 'MP' is mass production stage which is finally released to customers, 'OTA' is Over the Air releases for future enhancements or maintenance of software.

Most important challenge for any software product is "reduce defects delivered to customers". Every release made to customers have goals: how many more features delivered with lesser defects from previous release. Evelyn [15] study focuses on how customer defects can be analyzed systematically and gathered information and actions can be incorporated into testing. Her findings were indicating that most of defects were based on how existing or old features were used in different environments and configurations whereas most of testing focus was on new features.

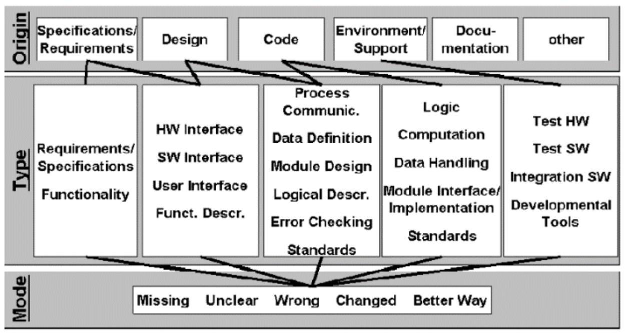
Every software released to customers depend on increasing demand of new features and existing defects being fixed. This is being catered based on software development practices which focus on faster delivery of new features incorporated without sacrificing quality. Factors which are most crucial for software development process are: interval, quality and cost; Marek, Dewayne and Dieter [7]. If software defects are caught later stages of development or software milestones fixing them becomes costly affair. So, knowing kind of defects, what are root cause of defects and taking countermeasures to detect them early and repair them is very critical. Software deliverables high level goals are: shorter interval to deliver features and fix bugs, higher quality and lesser or no defects and lowest cost. Defect root causes analysis is very critical as multiple defects can be detected due to same root cause and might be missed while design, implementation and testing.

Basically, there are two kinds of cause for defects being introduced: one is technology related – technical defects and another is procedure related – procedural or process related. The defects are caught while doing ‘design & implementation’ review and testing. Root cause analysis must be done to understands these aspects and accordingly corrective actions must be planned.

Number of defects detected is a one of important ways to measure software quality. It is one of the metric of software quality. At every phase of development and testing process how many defects detected determines software quality in general. But, that is not unified ways as every defect might be different priority and severity based on impact. Defects impact based on severity and priority defines majorly what is quality in terms of end user’s perspectives. Origin of defect or the phase where defect was introduced is one of another metric to define software quality. The phase where defects originates defines its impact.

There are two industry standard schemes: one is HP scheme and another is Orthogonal Defects Classification (ODC) Scheme.

The HP Scheme talks about three attributes: A. Origin – where defect was introduced, B. Type – describes defect of a particular origin in more detail and C. Mode – describes why defect was a defect. Can be represented as:



ODC scheme explains two perspectives of defects. One is: when a defect is opened, what are circumstances which leads to defect and what is impact to the user. Another is: when defect is fixed what is nature of defect and what is scope of the fix. In this scheme, mainly three attributes are collected when defect is opened: Activity, Trigger and Impact. Also in this scheme five attributes are collected when defect is closed: target, Defect Type, Qualifier, Age and Source.

Overall, it could be said that “Quality is directly associated to defects and defects leads towards improving quality eventually and continuously”.

3.2 Scope of Study:

1. To design and implement a model where "systematic corrective actions" can be guaranteed to prevent any recurrence of issues happening in the field which impact the quality. This systematic corrective action model will be implemented at all stages of SW flow viz. Requirements, Design, Implementations, Verification and Maintenance.
2. In current SW system environments, there are no specialized process or practices which can associate how quality is directly associated to "Root cause and Corrective Actions" of a defect, it's type, priority and severity. Fewer the issues reported by end users better is the quality of SW systems.

3.3 Objectives of this Study:

1. Create a Model based on Root Cause & Corrective Actions to help improve Quality of a Software System:
   A. Classifying each software systems into different major categories
   B. Quantifying the Q-Score of each system
   C. Defining major categories of root causes based on root cause analysis (RCA) of issues
   D. Defining Q-Score weightage associated with each categories of root causes
   E. Defining systematic preventive and corrective actions associated at every stage of SW flow
2. Based on this process model at each stage of SW system test plans can be defined ensuring issues or recurrence of issues are not happening, helping more tests to be passed to help improve Q-Score or pass % of the system. Eventually ensuring lesser defects delivered to customers.
3. Generalize a model where Quality is not measured only based on Pass% or how many tests executed and how many passed. Rather take SW phase, defects origin, it's type, priority, severity and test types also in create pass% weightage.

3.4 Related Work:

Marek, Dewayne and Dieter [7] did a case study in Root cause defect analysis study (RCA) of a project having goals as:

- Analyze sample defect modification requests (MRs) and find systematic root cause of defects
- Analyze major customer reported modification requests during maintenance release (called post-GA MRs, GA = general availability of the product)
- Proposed improvement actions as input for current development project to reduce number of critical defects and reduce defect fix effort.

Here study was made based on modification requests data taken from MR database and have MR classification scheme. In this study Marek, Dewayne and Dieter [7] explain MR classification schemes as: process phase where defect is found, classes of defects, defects types, defect nature, defects severity, defect location, defect triggers (root cause). Four root cause dimensions discussed are: phase triggers, human triggers, project triggers and review triggers. In this approach, rather than only one-dimensional root cause classification (single unique root cause) it was suggested to use four-dimensional root cause space. Basically, here strategy was to define countermeasures based on root causes and implement improvement actions effectively.

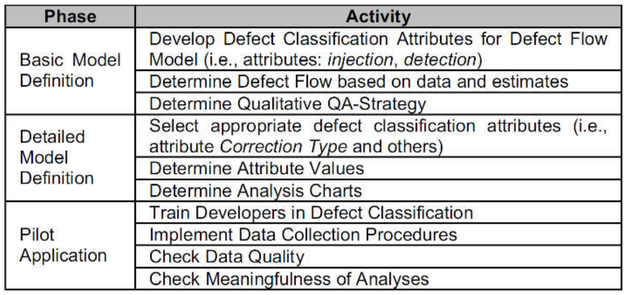
Evelyn [15] stresses that numerous study was made on how to catch defects early based improvements made in areas of requirements, design, code review, implementations, etc. At same time, there must also be efforts and continuous improvements in area of "testing techniques" based on defects which were detected in the field. She explains that just adding a test case to catch customer defect is not effective solution. Chances of same test case failing again based on same sequence of events are very low. Her analysis is mostly based on Orthogonal Defect Classification (ODC) [19] to use triggers – environment or conditions in which defect is exposed. The key difference in her study from ODC is termed as: "Minimum Conditions". Minimum conditions depict the essential minimum perquisites for the defect to occur. Minimal conditions help test engineers to create an environment or testing techniques to help repro problems efficiently. Her study suggests that most important reasons for defects are: due to change in environment or configuration causing existing features to break and change in code due to defects being fixed. This study tells that based on minimum conditions – test environments can be set, test cases chosen, exploratory testing and automated testing defined.

Naomi and Shigeru [12] talks about to improve software quality it needs to understand root causes of the defects. If root cause is understood by using technique "why-why analysis" fixing that will help close all defects with same root cause. They describe "defect root cause analysis and 1+n procedure" on how to improve software quality. This study is basically based on root cause analysis techniques, mainly: 1. Why analysis – five times why – also termed as "Why-Why Analysis" 2. FTA – fault tree analysis and 3. FMEA – Failure Mode Effect Analysis. FTA & FMEA are basically techniques designed to analyze causes of a predicted software failure mode.

Why-why analysis basically suggests adding tests into coverage based on root cause to improve quality. These additional tests are called "lateral search of defects. For any defect, there might be multiple root causes called 'Specific Root Cause' or 'Common Root Cause'. Specific root cause is direct root cause of the defect and common root cause is indirect cause of the defect. Once specific root cause is fixed defect will not occur again but if only common root cause is fixed there is still probability that defect might occur. Defect trend assessment, defect root cause analysis and 1+n procedure and defect convergence determination are important steps in this study. In this study three kinds of root causes are focused: cause of defect introduction in design, cause of defect overlooked in design review and cause of defect overlooked in testing. In this study, the development process model is taken in V model. So, basically this study, "defect root cause analysis" analyzes these three types of specific root causes and "1+n procedure" detects defects of same kind. 1 represents original defect analyzed and n represents total number of same kind defects detected.
This study also points out that since "why-why analysis" is based on 'specific root cause' and 'common root cause' analysis it's inefficient to catch same kind of defects. This study solves this problem and effectively can detect defects of same kind by narrowing down only on "specific root cause".

Bernd, Christian & Markus [16] present an approach they developed to define, introduce, and validate a customized defect classification scheme. They discussed how quality management is associated with defect classification schemes. They find that Hewlett-Packard (HP) Scheme and IBM's the Orthogonal

Defect Classification (ODC) schemes are too generalized to be incorporated for every companies. They discuss the need of customized defect classification schemes. The most fundamental approach is to combine “software engineering know-how of measurement experts” e.g. principles of defect classification schemes (successful elements of existing schemes) and “the domain know-how of developers” (knowing essentials of defects and special context of the project working on). In their study development process was customized as: Requirements, Concept Development, Concept Discussion, Function Specification, Function Review, Code Implementation, Code Review, Functional Test, Integration, Integration Test and Calibration. Quality assurance process is followed throughout development process in this model. Here quality measurement is based on techniques of finding which defects are introduced in which development activity and with which quality assurance techniques they are detected and fixed. In this study, the proposed process is based on interviews prepared by measurement experts and in which knowledge and experience is captured of domain experts. Whole process in a tabular form can be understood as below:



Phase	Activity
Basic Model Definition	Develop Defect Classification Attributes for Defect Flow Model (i.e., attributes: *injection*, *detection*)
	Determine Defect Flow based on data and estimates
	Determine Qualitative QA-Strategy
Detailed Model Definition	Select appropriate defect classification attributes (i.e., attribute *Correction Type* and others)
	Determine Attribute Values
	Determine Analysis Charts
Pilot Application	Train Developers in Defect Classification
	Implement Data Collection Procedures
	Check Data Quality
	Check Meaningfulness of Analyses

3.5 Limitation to Other Works:

Marek, Dewayne and Dieter [7] basically did study based on defects limited to product under study, defined four-dimensional root cause space and proposed countermeasures and improvement actions. Here it’s not discussed basically what is current state of quality, how it’s measured and based on countermeasures and improvement actions what is improvement in quality of software system. There are limitations in terms of how this process can be generalized and used for different products and organizations to help improve software quality.

Evelyn [15] mostly discussed about: process improvements based on minimum conditions, testing strategy improvements, exploratory testing, regression testing, system testing and more automated testing based on defects found by customers. It majorly focuses on testing existing features based on changed environments or configurations. Quality improvements depicts mainly how next release customer found defects (CFDs) are reduced.

Naomi and Shigeru [12] discussed “why-why analysis” focuses on both ‘Specific root cause’ and ‘common root cause’. Their techniques ”root cause analysis & 1+n procedure” only focuses on specific root cause of the defects. Why-why analysis is extensively applicable while 1+n procedure is extremely objective-oriented. It is based on only V software development model. It is only suitable to point specific root cause of a problem and fix that so it’s not suitable for wide range of problems. Since it takes lots of time to root cause of a problem, this technique limits to very important problem.
Another limitation here in this study is that the person who generated and introduced the defects is only responsible for root cause analysis. “1+n procedure” needs to keep repeating till all defects of same kind are not detected. In this technique quality is measured against prior and after the process how many defects of same kind is getting detected and % defects are still getting delivered.

Bernd, Christian & Markus [16] mostly have specialized classification scheme based on automotive embedded systems. They mainly depend on defect types defined by HP Scheme & ODC scheme. Their major focus is mostly to define defect types and determine which quality assurance technique can address which types of defect. In this study the dependency is mainly on structured interviews conducted with developers and their knowledge.

4. Methodologies:

It will be combination of:

A. Causal research (Experiment) – here Quality Score (Q-Score) is dependent variable. The possible independent variables will be based on: Phase of Development Cycles, Defect Types, Root Cause of defects and Test types
B. Descriptive research (Survey) – will have key aspects as below:
   i. Questionnaire based on problem & variables under study
   ii. Purposive Sampling – Pick N large scale software based companies mainly located in Pune. Each company's "Quality Head" who owns the quality of software deliverables in these software organizations will be asked for feedback

5. Conclusion:

This paper studies aspects of software quality, defects, root cause and corrective actions of defects. In this study current software quality model and various related work is analyzed. Current software quality model is mainly based on how many tests are executed and how many tests passed and failed. There are many other aspects of quality approach is discussed based on Root Cause Analysis (RCA). In this study relevant RCA methods are analyzed how they help improve quality of a software system.

This paper study focuses on feasibility analysis of creating a generalized quality model. This quality model will be based on mainly defects, root cause of defects, corrective actions of defects and other major practices and aspects of software development and testing techniques.

6. Limitations and future work:

1. Whole study will be based on defects study of Android based embedded system software product. Defects under analysis are from May 2013 to April 2017.
2. Study is based on major embedded software companies located only in Pune, India.
3. Root cause and corrective actions action are completely manual process in this study. Later part of study can be focused on how defects are analyzed automatically, co-relate them to the changes made in design, implementation, testing and bug fixes generating the defect. Automatically modelling of what are changes based tests coverage needed, automatically list down test coverage and trigger the tests to avoid regressions and defects. These automatic actions can be taken based on machine learning and artificial intelligence techniques.

7. Acknowledgement

A sincere thanks to Prof. (Dr.) R. Kamatchi, Professor, Head, CSE, Amity School of Engineering and Technology, Mumbai; being mentor and guide and for continuous help and support to come up this review paper.

8. References:

"Principles and rules are intended to provide a thinking man with a frame of reference."

1. **Roger Pressman**; Software Engineering – A Practitioner's Approach; Tata McGraw Hill, Sixth Edition, 2010

2. **Pankaj Jalote's** Software Engineering – A Precise Approach, Wiley Precise, First Edition, 2010

3. 'Measuring Application Development Productivity' by **Allan Albrecht** at IBM, 1979 http://www.bfpug.com.br/Artigos/Albrecht/MeasuringApplicationDevelopmentProductivity.pdf

4. "Software Quality in 2012: A Survey of the State of the Art" by **Capers Jones**, Vice President and Chief Technology Officer, Namcook Analytics LLC, May 1, 2012, www.Namcook.com http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf

5. **Jones, Capers & Bonsignour, Olivier**; The Economics of Software Quality;
Addison Wesley, 2011
http://ptgmedia.pearsoncmg.com/images/9780132582209/samplepages/0132582201.pdf

6. **Kan, Steve**; Metrics and Models in Software Quality Engineering, Addison Wesley, 2003 http://hornad.fei.tuke.sk/kpi/person/samuelis/metrics/Metrics%20and%20Models%20in%20Software%20Quality%20Engineering,%202ed%20%5BAddison%20Wesley'2002%5D.pdf

7. **Marek Leszak**; Lucent Technologies, Optical Networking Group, Thurn-und-Taxis-Str. 10, 90411 Nuernberg, Germany. **Dewayne E. Perry**; Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX. **Dieter Stoll**; Lucent Technologies, Optical Networking Group, Thurn-und-Taxis-Str. 10, 90411 Nuernberg, Germany: A case study in root cause defect analysis
Proceeding. ICSE '00 Proceedings of the 22nd international conference on Software engineering Pages 428-437 ACM New York, NY, USA ©2000 table of contents ISBN:1-58113-206-9 doi>10.1145/337180.337232

8. **Mikko Nieminen**; Digital Syst. & Services, VTT Tech. Res. Centre of Finland, Oulu, Finland. **Tomi Räty**; Digital Syst. & Services, VTT Tech. Res. Centre of Finland, Oulu, Finland: Adaptable Design for Root Cause Analysis of a Model-Based Software Testing Process. Information Technology - New Generations (ITNG), 2015 12th International Conference on. 978-1-4799-8828-0/15 $31.00 © 2015 IEEE DOI 10.1109/ITNG.2015.67

9. **Timo O. A. Lehtinen**; Sch. of Sci., Dept. of Comput. Sci. & Eng., Aalto Univ., Espoo, Finland. **Mika V. Mantyla**; Sch. of Sci., Dept. of Comput. Sci. & Eng., Aalto Univ., Espoo, Finland. What Are Problem Causes of Software Projects? Data of Root Cause Analysis at Four Software Companies. 2011 International Symposium on Empirical Software Engineering and Measurement. 978-0-7695-4604-9/11 $26.00 © 2011 IEEE DOI 10.1109/ESEM.2011.55

10. **Ferdian Thung; David Lo; Lingxiao Jiang**; Sch. of Inf. Syst., Singapore Manage. Univ., Singapore, Singapore. Automatic recovery of root causes from bug-fixing changes. 2013 20th Working Conference on Reverse Engineering (WCRE). 978-1-4799-2931-3/13/$31.00 c 2013 IEEE

11. **Li Meng ; Xiaoyuan He**; Neusoft Group Ltd, China.
Ashok Sontakke; Nihilent Technologies Pvt. Ltd., Pune, 411001, India. Defect Prevention: A General Framework and Its Application. 2006 Sixth International Conference on Quality Software (QSIC'06) 0-7695-2718-3/06 $20.00 © 2006, IEEE.

12. **Naomi Honda;** IT Software Operations Unit, NEC Corporation, 1-10, Nisshin-cho, Fuchu, Tokyo 183-8501, Japan e-mail: n-honda@ay.jp.nec.com.
Shigeru Yamada. Department of Social Management Engineering, Graduate School of Engineering, Tottori University, Tottori-shi 680-8552, Japan e-mail: yamada@sse.tottori-u.ac.jp. - ''Defect Root-Cause Analysis and 1+n Procedure'' technique to improve software quality. Received: 3 April 2012 / Revised: 20 June 2012 / Published online: 25 July 2012 The Society for Reliability Engineering, Quality and Operations Management (SREQOM), India and The Division of Operation and Maintenance, Lulea University of Technology, Sweden 2012. Springer.

13. **Cagla Atagoren**; Information Technology and System Management, Bas¸kent University, 06810, Ankara, Turkey e-mail: caglaatagoren@gmail.com. **Oumout Chouseinoglou**; Statistics and Computer Science Department, Bas¸kent University, 06810, Ankara, Turkey e-mail: umuth@baskent.edu.tr. A Case Study in Defect Measurement and Root Cause Analysis in a Turkish Software Organization. R. Lee (Ed.): SERA, SCI 496, pp. 55–72. DOI: 10.1007/978-3-319-00948-3_4 c Springer International Publishing Switzerland 2014

14. **Fuping ZENG, Aizhen CHEN, Xin TAO**; Department of System Engineering Beihang University Beijing, 100191, China. Study on Software Reliability Design Criteria Based on Defect Patterns. 978-1-4244-4905-7/09/$25.00©2009 IEEE

15. **Evelyn Moritz**; AVAYA emoritz@avaya.com. Case Study: How Analysis of Customer Found Defects Can Be Used by System Test to Improve Quality. Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on. ICSE'09, May 16-24, 2009, Vancouver, Canada 978-1-4244-3494-7/09/$25.00 © 2009 IEEE

16. **Bernd Freimut**; Fraunhofer IESE; Sauerwiesen 6; 67661 Kaiserslautern, Germany; freimut@iese.fhg.de. **Christian Denge**; Fraunhofer IESE; Sauerwiesen 6; 67661 Kaiserslautern, Germany; denger@iese.fhg.de. **Markus Ketterer**; Bosch GS-EC/ESP; P.O. Box 30 02 40; 70442 Stuttgart, Germany; Markus.Ketterer@de.bosch.com. An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management. 11th IEEE International Software Metrics Symposium (METRICS'05).

17. **Marcos Kalinowski, Guilherme H. Travassos, and David N. Card**; COPPE – Federal University of Rio de Janeiro – Brazil, 2 Bennett – Methodist University of Rio de Janeiro – Brazil, 3 Det Norske Veritas - USA [mkali, ght]@cos.ufrj.br , card@computer.org. Towards a Defect Prevention Based Process Improvement Approach. 2008 34th Euromicro Conference Software Engineering and Advanced

Applications. 978-0-7695-3276-9/08 $25.00 © 2008 IEEE DOI 10.1109/SEAA.2008.47.

18. **J.H. van Moll**; Philips Semiconductors ReUse Technology Group Prof. Holstlaan 4 5656 AA Eindhoven The Netherlands jan.van.moll@philips.com. **J. C. Jacobs;** Philips Semiconductors ReUse Technology Group Prof. Holstlaan 4 5656 AA Eindhoven The Netherlands jef.jacobs@philips.com. B. **Freimut Fraunhofer**; IESE Sauerwiesen 6 67661 Kaiserslautern Germany freimut@iese.fhg.de. J.J.M. **Trienekens Frits Philips**; Institute Eindhoven University of Technology Den Dolech 2 5600 MB Eindhoven The Netherlands j.j.m.trienekens@tm.tue.nl. The Importance of Life Cycle Modeling to Defect Detection and Prevention. Software Technology and Engineering Practice, 2002. STEP 2002. Proceedings. 10th International Workshop on. Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02) 0-7695-1878-8/02 $17.00 © 2002 IEEE.

19. M. Butcher, H. Munro & M. Butcher, "Improving software testing via ODC: Three case studies", IBM Systems Journal, Vol. 41, No. 1, pp.31-44 (2002). https://researcher.watson.ibm.com/researcher/view_group.php?id=480