Spring 5-18-2019

Long Term Software Quality and Reliability Assurance in a Small Company

Eric Abuta
eabuta@gmail.com

LONG TERM SOFTWARE QUALITY AND RELIABILITY
ASSURANCE IN A SMALL COMPANY

Approved by:

Dr. Jeff Tian

Dr. Sukumaran Nair

Dr. LiGuo Huang

Dr. Jennifer Dworak

Dr. Jimmy Hosch

LONG TERM SOFTWARE QUALITY AND RELIABILITY ASSURANCE IN A SMALL COMPANY

A Praxis Presented to the Graduate Faculty of the

Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Engineering

with a

Major in Software Engineering

by

Eric N. Abuta

(B.S. Computer Science, Texas Tech University, 1996)
(M.S., Southern Methodist University, 2001)

May, 2019

ACKNOWLEDGMENTS

I want to first thank my family and friends for giving me the support through almost 10 year journey working on this Doctor of Engineering (DE) program. I want to particularly thank my wife Patricia Ogari for the emotional support and picking extra family related duties in order for me to concentrate on my school work. I also want to thank my children, Marita and Shali for their challenge I graduate before they finish high school. My parents Joseph and Agenta Abuta for sending me to a foreign country to pursue my study of interest. My siblings Anthony, Edward, Robert and Janet Abuta for always being there. And finally all my soccer teammates, particular the TI Rebels team's encouragement for me to stay the course.

I also want to acknowledge Mike Whelan, Owner and President of Verity Instruments for allowing me to use the Optical Endpoint Detection (OED) software system data for my study. Barbara Railey of Verity instrument for her honest and strict participation in proctoring all my exams over the years. My supervisor Rick Daignault for his support and recommendation for me to join the DE program. Dr. John Corless of Verity Instruments for his help with R tool. Mr. John Hirsh of Verity Instruments for his help mining all historical data from the defect tracking reporting system. Dr. Jimmy Hosch for his intellectual discussions and recommendation for me to join the DE program.

Abuta , Eric N. B.S. Computer Science, Texas Tech University, 1996
M.S., Southern Methodist University, 2001

Long Term Software Quality and Reliability
Assurance in a Small Company

Advisor: Professor Jeff Tian
Doctor of Engineering degree conferred May, 2019
Praxis completed December, 2018

Demonstrating software reliability across multiple software releases has become essential in making informed decisions of upgrading software releases without impacting significantly end users' characterized processes and software quality standards. Standard defect and workload data normally collected in a typical small software development organization can be used for this purpose. Most of these organizations are normally under aggressive schedules with limited resources and data availability that are significantly different from large commercial software organizations where software reliability engineering has been successfully applied.

The objective of this study on a Semiconductor Optical Endpoint Detection (OED) software system was to demonstrate how to measure software reliability in multiple releases and whether continuous defect fixes and code upgrades increased software reliability. The defect data used for this study was reported from August 2001 through July 2014 and was organized based on the whole software as an entity of its own and branches based on code freezes and releases at different times of the calendar period under each branch.

This study looked at techniques such as trend test that evaluated OED overall trend and stability, input domain reliability models (IDRM) that assessed the system's operational reliability, software reliability growth models (SRGM) that tracked the system's reliability growth, and orthogonal defect classification (ODC) that provided

in-process feedback for focused defect removal and quality improvement.

Laplace trend test was used on defects over years and a general trend of improved reliability across the board was observed. This was followed by assessing the success rate or operational reliability per Nelson's IDRM for each given month. The data clearly showed an initial unstable period and then a prolonged stable period of all the software entities and branches observed. Goel-Okumoto (GO), Musa-Okumoto (MO), and Yamada SRGMs were then fitted to the defect data. It was observed that the model trends followed closely to the actual raw defect data and provided quantitative evidence of reliability growth.

To bridge the gap between defect and reliability analysis, ODC was considered to be an appropriate approach to achieve this objective. This study adapted the original ODC and demonstrated the ability to provide in-process feedback for a semiconductor software using data that is normally found in a small company. To assess the impact of this in-process feedback, the study first quantified the baseline defect count, distribution, and reliability growth. The study then quantified the quality improvement using the same metrics after actions resulted from in-process feedback. The comparison results demonstrated that the adapted ODC offered valuable early in-process feedback that led to quantifiable improvement.

In conclusion, this study initially analyzed data using Laplace trend test results to give a quick assessment and visualization of the reliability growth trend. Then this study used IDRMs operational reliability analysis results to provide a stable and robust reliability estimates for an extended stable period after some initial fluctuation. This study then took the reliability growth analysis results by SRGMs as evidence that continuous defect fixes increased software reliability substantially over time. Finally, this study concluded by demonstrating that ODC can offer valuable early in-process feedback using defect count, defect distribution, and reliability growth metrics to

quantify improvement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

To Patricia Ogari, Tatyana Marita Abuta, and Shali Nyang'ara Abuta

Chapter 1

INTRODUCTION

Software reliability is the probability of a software system to perform its function failure-free during a specified time on a given set of inputs under defined conditions [54][68]. Software reliability can also be defined in terms of trend of reduced defects discovered over time through defect analysis and classification. This study demonstrates software reliability across consecutive software releases through defect analysis, trend, and classification for a semiconductor Optical Endpoint Detection (OED) software system using standard defect data that is normally available to many software development organizations. The OED is connected to an optical sensing instrument that transmits optical data to be processed by OED's proprietary algorithms.

There is general recognition of a need to measure and analyze software reliability between releases. This can be achieved by documenting reliability of OED using historical data to predict future reliability trend. Stakeholders can also use this data as an early indicator of quality issues if actual data is not conforming to expected and predicted trend. In turn the necessary counter measures can be applied to improve quality. This information can also assist some of the end users to make informed decisions on software upgrades. Since most of the users have very restricted process of upgrading their software, putting reported defects in perspective and showing trend on reliability and stability would be very instrumental to their change control boards.

This Praxis is divided into several chapters, starting off with this chapter on a general introduction and layout of this Praxis. Chapter 2 discusses related work that includes, software quality and reliability, defect management and analysis, orthogonal

defect classification (ODC), software reliability, and Operational Profile (OP).

The Praxis then transitions in Chapter 3 to address the problem statement, OED's environment, system flow, data, testing, test effort tracking, and solution strategy. In Chapter 4, the Praxis then discusses OED's defect analysis, focusing on failure distribution and data visualization, cumulative defects over time, unfixed defects over time, trend testing using Laplace, and defects over test runs trend analysis.

Chapter 5 addresses OED's reliability, looking at operational reliability and reliability growth, concluding Chapter 5 with a discussion on reliability prediction, specifically prediction based on 50% and 75% training based data analysis.

The Praxis then looks at the overall ODC in Chapter 6 including scaling down and adapting of applicable original ODC attributes for OED. The chapter then covers baseline analysis on metrics to be used for post ODC analysis and concludes the chapter through validation of the ODC data. The Praxis finally summarizes this study in Chapter 7 and draws conclusions.

Chapter 2

RELATED WORK

A well-defined system and mechanism is needed in order to record, categorize, track, and analyze defects and reliability of a given system. This chapter will go over previously studied work on the main areas this study will cover. The chapter will first go over general software quality and reliability, then look at the defect management and analysis area, followed by Orthogonal Defect classification (ODC) and finally discussing software reliability models.

2.1. Software Quality and Reliability

Before defining what software quality is, it would be first ideal to discuss what the definition of quality is [4][17][22][40]. According to David Garvin study [26], quality can be viewed in five perspectives, transcendental, user, manufacturing, product, and value-based view.

In transcendental view, quality is something that can be recognized by an individual through a delight experience but difficult to define. Therefore, quality would vary from user to user based on their experience. Objective of quality in the user view is to meet the user's needs. Quality is more definite compared to in the transcendental view. In the manufacturing view, quality is centered on the product itself during manufacturing, conforming to established processes. In the product view, quality is centered on the actual product's attributes and characteristics. This has led to models to be developed to tie between the product view and user view. Finally, in the value-base view, quality is purely based on the customer's willingness to pay for the

product after trade-offs are considered between cost and quality [46].

Based on the above definition, one can define software quality as simply the user expectation of the software performing correctly per its specification. The system meets all promises while limiting negative impact such as disruption or serious looses, especially financially. Subsequently delighting the end user to a point they can even promote the system [23][80].

Reliability is the probability of a system performing its function during a specified time and under defined conditions. Software is considered dependable when it is reliable, available, operates correctly, and its data secure from unauthorized operators [7][37][66].

In software engineering discipline, quality assurance covers multiple aspects including but not limiting to the functionality, schedule, cost, and reliability. For functionality, automation replaces previously manual processes while schedule accounts for scheduling critical features and systems to be incorporated into a system on a timely manner to address end user's needs. In cost aspect, keeping cost at a minimum would aid in staying competitive while reliability would lead to managing end users' quality expectations as software systems have become increasingly depended upon [14][28][36][66][81].

Software assurance is the confidence level the software will function as expected without vulnerabilities [24]. Software improvement is the process of constantly involving and communicating with end users to ensure the software meets their expectations and to remove any pending hindrances to achieving their end objectives [74].

2.2. Defect management and analysis

A defect normally is a reference to a failure or a fault [55]. A failure is a performance deviation of a software system from its required or expected function. A fault is

the wrong process, step or data definition in a computer program [25][30][33][41][75]. Defect management process involves clearly defining defect lifetime management, role of users of the defect management system, clear boundary of defects between multiple products and functionality, a cross-reference for the defects, and general distribution and root cause analysis [47][48][34].

There are multiple mechanisms for defect analysis that include defect prediction between software versions and trend analysis [50][54][32]. Since the data used for analyzing reliability is based on failure in some sort of an interval, Laplace test, for example, can be a viable option [42]. The Laplace factor l(t) is determined by:

$$l(t) = \frac{\frac{1}{i-1}\sum_{n=1}^{i-1}\sum_{j-1}^{n} t_j - \frac{\sum_{j=1}^{i} t_j}{2}}{\sqrt[t]{\frac{1}{12(i-1)}}} \tag{2.1}$$

When the Laplace factor is negative it can be interpreted as decreasing failure intensity which implies reliability growth or most defects have been found. When the factor is positive it can be interpreted as increased failure intensity, thus, reliability decay [43]. This technique was chosen as it seemed appropriate for the data set available for this study.

2.3. Orthogonal Defect Classification (ODC)

ODC technique affords in-process feedback to stakeholders such as developers and testers of a software through defects classification and analysis [13][77]. ODC acts as a bridge between casual analysis and statistical defect models. Causal analysis identifies root cause of the defects mostly by knowledgeable personnel [20]. One of its drawbacks is the limitation of human ability for such analysis. Statistical defects typically involves reliability growth that has some prediction that requires time and feedback to come much later in the process. This does not benefit the developers

earlier in the development process [15][18][19].

In ODC, defects are classified into orthogonal attributes that can be sifted through in order to arrive at a pattern [27]. First level of ODC involves classification of the defects under different types such as requirements, design, logical and documentation. Once classified, the defects are then analyzed. This analysis is referred to as Root Cause Analysis (RCA) [61]. The goal of RCA is to identify the root cause of the defects and trigger corrective action to eliminate the source of the defects.

Analysis of the defects under ODC can include one-way analysis where by one attribute is analyzed at a time, while a two-way or a multiple-way analysis involves analysis carried out on the interaction of two or more attributes. ODC technique identifies defects by grouping them into two major categories, the opener and closer sections. The opener section refers to the time when a defect was first detected. The closer section refers to the time when a defect gets resolved [21][52][11].

The opener section typically has three ODC attributes; activity, trigger, and impact [72]. Under the activity attribute, design review and code inspection, as an example, can be considered as activities. The trigger attribute can be defined under each activity attribute. Going by the activity attribute discussed above, the trigger attributes for design review and code inspection attributes are more or less similar and could range from design conformance, logic, to backward compatibility and rare situations. In the case of the unit test activity attribute, there could be simple or complex path trigger attributes. While for the function test activity there could be test coverage, variation, sequence, and interaction triggers. Finally for the system test activity there could be stress, recovery, start up, hardware configuration, and software configuration blocked test triggers.

The impact attribute presents the opportunity to ask how the defect can impact the end user. Generically speaking, the following are the potential impacts under

Table 2.1. Opener Section Attributes

Activity	Triggers	Impact
Design Review	Design Conformance	Installable
Code Inspection	Logic or flow	Serviceability
Unit Test	Backward Compatibility	Standards
Function Test	Lateral Compatibility	Integrity or security
System Test	Concurrency	Migration
	Internal Document	Reliability
	Language Dependency	Performance
	Side Effect	Documentation
	Rare Situations	Requirements
	Simple Path	Maintenance
	Complex Path	Usability
	Test Coverage	Accessibility
	Test Variation	
	Test Sequencing	
	Test Interaction	
	Workload or Stress	
	Recovery or Exception	
	Start up or Restart	
	Hardware Configuration	
	Software Configuration	
	Blocked Test	

this attribute that include but are not limited to the ability to install the system, integrity and security, performance, maintainability, serviceability, ability to migrate the system with limited impact, easy of documenting, usability, reliability, meeting requirements, and accessibility just to mention a few [12][16][21][52]. Table 2.1 summaries the mapping of the opener section ODC attributes discussed above.

For the closer section, there are typically five attributes that include target, defect type, qualifier, age and source attributes. The target attribute has entries such as design and code, while the defect type attribute centers around the correction made to address the defect. The age attribute looks at how long it has been since the defect was discovered while the source attribute would best define the fix in terms of either

Table 2.2. Opener Section Attribute

Target	Defect type	Qualifier	Age	Source
Design	Initialization	Missing	Base	In-House Coding
Code	Checking	Incorrect	New	Third Party Library
	Algorithms	Extraneous	Rewritten	Outsource
	Functions		ReFixed	Ported
	Timing			
	Interface Messages			
	Relationship			

requirements, design, code, build packaging, information development, and national language support.

The defect type attribute is centered around the correction made to address the defect. The defect type target could be on the design or code, looking specifically at assignment or initialization, checking for invalid inputs, algorithms correctness, functions of code, timing, interface messaging just to mention a few.

The qualifier attribute looks at the missing or inapplicable implementation, example using incorrect information. The source attribute would best define the fix in terms of either requirements, design, or code. Some aspects to be looked into could be in house development verses third part libraries or outsourcing custom development.

The age attribute looks at how long the defect was. Whether it was in the original base code, new defect or rewritten defects [11][12][16][21]. Table 2.2 summarizes the above discussed closer attributes.

Initially developed at IBM [21], ODC has successfully been used in multiple industries in identifying problems with software quality and provided means of improving software quality [18][35][51]. The IBM study successfully demonstrated an in-process feedback that was beneficial to developers, providing product progress through defined process using defect type distribution measurement. Subsequent studies such

the one in NASA [53] adapted the original ODC and successfully demonstrated a combination of manual and automated machine learning analysis. In both cases, the studies had access to vast data typically found in larger software development organizations. Other studies also adapted ODC to look at in-process usability problems [27], improved reliability based on classifying and analyzing web related errors [56], and defect analysis on a Software as a Service (SaaS) in the cloud [6][5][15]. In this study, we would demonstrate an adaption of ODC using limited data typically found in small development organizations.

2.4. Software Reliability

Software reliability measures the probability of failure-free software operation with given specific conditions and time or input [54][68]. Reliability models are tools used to assess, predict, and control software reliability[44][45][49][58][59]. Software reliability growth model (SRGM) and input domain reliability models (IDRM) are the commonly used types of reliability models. SRGMs use a time domain approach to determine the probability of failure as a function of the number of faults over time. On the other hand, IDRMs analyze input states and failure data. The specific models used in this study were determined based on, among other factors, recommended standards [38], the simplicity of the models, and how widely the models have been used in the past [8][64][78][79].

SRGMs include concave and S-shaped models such as Goel-Okumoto (GO) and Yamada respectively [31][82]. These are a type of Non Homogenous Poisson Process (NHPP) models [55][68] where expected number of defects at time t are denoted as $\mu(t)$ with:

$$\mu(t) = a(1 - e^{-bt}) \tag{2.2}$$

for GO model, and

$$\mu(t) = a(1 - (1 + bt)e^{-bt}) \tag{2.3}$$

for the S-shape model, where a is the expected total number of defects and b is the shape factor.

Another variation of NHPP model is the Musa-Okumoto (MO) [69] that is based on logarithmic execution time represented by:

$$\mu(t) = \frac{1}{\theta} \ln(\lambda_0 \theta_t + 1) \tag{2.4}$$

where t is time, λ_0 is the initial failure intensity, and θ is the model parameter.

For this study, the non linear regressions R^2 [64] was used to determine the goodness of fit for the selected SRGMs. R^2 value is between 0 and 1, with R^2 close to 1 being considered as a good fit.

Purification level ρ can be used to evaluate reliability growth quantitatively. ρ is the ratio of the failure rate reduction over a given test period define by:

$$\rho = \frac{\lambda_0 - \lambda_T}{\lambda_0} = 1 - \frac{\lambda_T}{\lambda_0} \tag{2.5}$$

where λ_0 is the failure rate at the start of testing and λ_T is the failure rate at the end of testing.

For the Input Domain Reliability Model(IDRM), the Nelson model [70] was selected. The model gives estimated reliability (R) for different time segments. The reliability is given by:

$$R = 1 - \lambda = 1 - \frac{f}{n} = \frac{(n - f)}{n} \tag{2.6}$$

where f is the number of failures, λ the failure rate and n the number of runs.

2.5. Operational Profile (OP)

Operational Profile (OP) is a quantitative characterization of how the software will be used . A profile is considered to be an independent possibility (element) and its occurrence probability. In general, in developing an OP, one will need to find a customer profile, establish a user profile, define a system mode profile, and determine functional and operation profile [76]

Musa's OP typically includes customer profile, user profile, system mode profile, and functional profile [65][67][29]. A customer profile consists of a list of independent customers that use the software product. A customer can be defined as one or a group of users with a common objective of feature sets or usage model. This can also be viewed as a list of customer types and their associated probabilities. Normally the probability is determined based on the proportion of the time each customer type uses the software system.

In some instances the system user may be different from the software customer. To distinguish the difference when defining a user profile, a user can be considered as a group or institution that operates the system while a customer can be considered as the entity that acquires the system. Therefore, user type can be defined as a set of users that can operate the system similarly with their associated probabilities. Just like the customer profile, the sales proportions of each user type will be based on the sales in a given period [29][65][67].

A system mode profile can be defined as the way a system operates. Independent segments of a system operation which can either be done sequentially or concurrently which share the same system resources. Therefore, for each system mode, an operational profile will need to be developed. The profiles will be a representation of a list of system modes and their corresponding occurrence probabilities [29][65][67].

A functional profile can be defined as explicit or implicit based on key input variables. Key input variables are external parameters which affect the operation of the software system. Based on the variation of the input variables, defined as levels, an explicit profile can involve each element to be designated by simultaneously specifying all levels of the key inputs needed for identification. On the other hand, an implicit profile can be expressed by sub profiles of each key variable. Probabilities are assigned based on the ranges that can legally be used [29][65][67].

Chapter 3

ENVIRONMENT, PROBLEMS, AND SOLUTION STRATEGY

This chapter will look at OED's flow, testing environment, and data. The discussion will go into what quality control measures were applied to ensure consistency and accuracy of the analysis. The discussion will then turn to the effort and test tracking activities. Then concluding with the solution strategy proposed to address the problems of this study.

3.1. Problem Statement

Before discussing the OED's flow, testing environment, and data, this section will first address the objective of this study. For sometime there was a need to quantify OED's reliability to its stake holders. A few defects reported from the field had ended up costing a few customers money, raising questions on OED's quality in the mind of these customers. Thus, questions were raised on what methodologies could be appropriate to use on OED to determine reliability, putting costly defects in perspective.

To initiate this study, a problem statement needed to be defined with a specific focus of importance on reliability across multiple releases. Thus, the problem statement was narrowed down to whether one can demonstrate the current software reliability, reliability growth over time, and understanding defects across multiple software releases using standard defect and workload data normally collected in a typical small software development organization. Whether continuous defect fixes increase software reliability substantially?

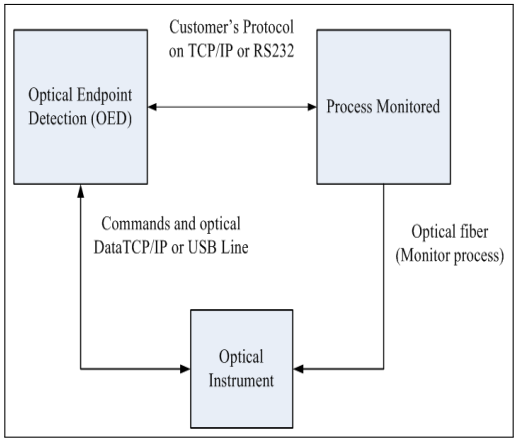

Figure 3.1. OED: System flow

3.2. OED System Flow and Testing Environment

Company ABC has taken measures to develop a process that can ensure certain information is collected and validated in the very limited and aggressive time constraints between development and deployment [1][3]. The process adapted for the OED ranges from requirements gathering, defects reporting, source code control to creating and maintaining a set of regressions test cases.

OED is an endpoint detection system that communicates to a process tool through a choice of the customer's communication method. OED sends commands to the instrument on how to collect and transmit data. The OED then runs the appropriate proprietary algorithm in the customer's recipe. When the condition to stop the process is reached, the OED sends appropriate command to the process tool to stop the process. An acknowledgment is sent to the OED which will then send the stop command to the instrument to stop collecting the data. Figure 3.1 illustrates the system described above.

OED development and operating environment presents unique challenges compared to large commercial or telecommunication environment where software reliabil-

ity engineering has been successfully applied [55][64][78]. Upgrades and deployments are end user controlled. This raises challenges on applying needed defect fixes unless compelling evidence is presented [1].

The feature sets were controlled through code freezes and branches. Once a targeted major feature was implemented and necessary verification carried out, a branch would be created. The branch was then maintained and only approved defects were addressed moving forward. Eventually sub branches were created from identified releases for specific customers and that set of code base was frozen. Any subsequent changes had to be approved by the customer [60][62]. All the software changes were controlled by a Software Change Control Board (SCCB) that determined what changes were to be made based on customer needs and company priorities [39][63][73].

OED end users in most cases will not provide any information of the system's performance due to proprietary claims. As a compromise, some end users will allow provision of actual error messages observed and in some cases some extracts of the failure related log files.

When defects are discovered, corresponding test cases are created to confirm they are addressed. If a test case is automatable, it is added to auto regressions test suite to be run in future software builds before deployment. For test cases that are not automatable, they are added to the manual regression suite that would only be run if the future code changes are considered to be at risk of breaking existing defect fix or features. This risk is determined by a proprietary methodology that looks at each module affected by the code changes and assigns a Software Change Impact Index (SCII). If a threshold is met, all the manual test cases under affected category will be run to ensure past feature sets are still functional.

3.3. OED Data, Categories and Quality Control

The data used for this study was based on the number of defects discovered, calendar date the defects were discovered, and number of test runs executed. All this information was stored and extracted from a defect reporting database system.

The defects under analysis were those that were on the released code base discovered either in the field or during in-house testing after deployment. In other words, after initial feature development, during formal test, the defects reported during this test period are not part of this analysis because of inconsistency of reporting. But once the software was deployed, any subsequent defects on this code base discovered either in the field or in-house were then tracked and analyzed in this paper. The test runs data was based on test cases generated and executed to verify new features and discovered defects.

The OED data was categorized based on calendar time and branches with the test interval of a month. The data in this study ranged from May 2001 to June 2014. The branches were denoted by the first digit of the release numbers. For example branch 4.X contained releases 4.01, 4.1, 4.2 etc. 4.X, 5.X and 6.X branches were deployed on September 2004, February 2007, and September 2011 respectively. All these branches are actively in use and supported. Table 3.1 highlights the Kilo-Source Lines of Code (KSLOC), the deployment date, and the number of releases from deployment date to June 2014.

The 2.X and 3.X branches support was discontinued and all users had been encouraged to migrate to later released branches. The 1.X branch was never deployed, but for the sake of this study, data from 1.X, 2.X, and 3.X branches was used to track defects of the whole OED system.

Table 3.1. Versions under each branch

Branch	KSLOC	Deployed	Number of Releases	First to Latest Version
3.X	477	October 2003	13	3.02 to 3.28
4.X	599	September 2004	14	4.01 to 4.42
5.X	848	February 2007	20	5.0.00 to 5.6.00-14
6.X	843	September 2011	10	6.0.00 to 6.4.03

3.4. Effort and Test Tracking

In general, the majority of the end users target a set of OED releases from one branch to a particular product line. For example, if tool X gets an OED release from the 4.X branch, all subsequent releases to tool X will get only releases from the 4.X branch. Very seldom is there a need for an OED release from another branch such as 5.X in this example. So for tool X, the reliability within the 4.X OED releases will be the only concern for this end user. If down the line this end user wants to release another tool Y, then depending on new features, a different OED branch can be adapted. Once the branch is identified and deployed, then all subsequent releases will only come from the same branch.

To evaluate the test effort applied to OED, the monthly test runs were plotted against the calendar time. Figure 3.2 has graphs from the mature 4.X and 5.X branches for evaluation. The first graph has the test runs over time plotted for the 4.X branch from the time period of January 2004 till September 2010; September 2010 was the last release made from this branch. Since there were minimum defects discovered, addressed, and tested after September 2010, the month was deemed appropriate point to end the evaluation time period.

The 5.X branch graph in Figure 3.2 also tended to take similar trend as observed in the 4.X branch. The initial effort of test runs drastically increase between January

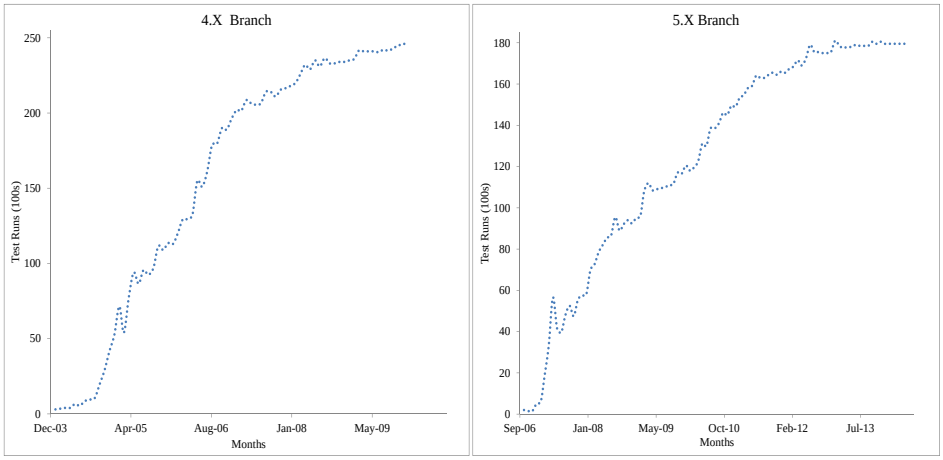

Figure 3.2. Test runs over time

and May 2007, indicating a lot of test effort was put to stabilize the branch. Then a steady increase of test runs was observed till July 2012 when the test runs trend started to taper off.

Both branch test run over time graphs seem to have similar trend where the test runs were progressively increasing as time progressed. More tests were run later to catch newer defects and account for the new features added to the system. Similarities between the branches show evidence that the test effort was applied consistently across the branches.

3.5. Solution Strategy

After the above discussion on OED data, categorization, and measures taken to ensure data consistency. The next logical step was to look at solution strategy on

answering the key question of this study. Mainly, determining reliability growth across releases. The solution was divided into defect analysis, orthogonal defect classification (ODC), and software reliability.

For defect analysis, cumulative defect over time analysis was taken, looking at the whole software entity trend and each branch specifically. These were uniquely defined defects and casual visualization was the primary approach of assessing viability of a reliability trend. Trend test was then applied using Laplace trend test on defects over years. Chapter 4 goes into a more detail discussion on defect analysis.

Once the defect analysis was carried out and reliability trend could be visualized, the next step was to look at reliability models using this derived data. Two types of reliability approach were considered. The first one was to look at the operational reliability using IDRM and the second to look at reliability growth using SRGMs. Chapter 5 goes over in depth on operation reliability and reliability growth respectively.

To bridge the gap between defect and reliability analysis, ODC was considered as an appropriate approach to achieve this objective. For a meaningful ODC analysis, a systematic scheme was developed in order to guide the analysis and to validate the results. This study's objective was to determine whether original ODC could successfully be adapted in a different environment and subsequently validate the results through defect count, defect distribution, and reliability growth metrics [2]. This study first considered and adapted original ODC attributes that were applicable. Secondly established an ODC baseline on defect count, defect distribution, and reliability growth metrics and came up with recommendations. Thirdly, compared and validated the baseline and post ODC results based on the three metrics mentioned above. This three-step solution strategy can be considered as an adaptation of the quality improvement paradigm [10] to achieve this study's research objective

mentioned above[9][71].

Historically, most testing activities had been left to testers, with very limited unit tests carried out by developers. This eventually started to show an unnecessary prolonged cycle of fixing defects and retest. The low lying fruit type of defects that could have been caught by a unit test ended up getting caught by the testers. In addition, unit tests had been proven to be vital in catching logic errors that were difficult to catch in the stand system test environment. Consequently, requirements for each developer to carry out unit test was implemented.

Going hand in hand with the unit and system test discovered defects, were defects discovered in the field by the customer. Due to the negative impact to customers and the limitation of accessing customer's proprietary operational profile, management was concerned about the relatively larger number of defects discovered in the field by customers. The fewer defects discovered by the end user, the fewer questions could be raised in the customer's mind regarding the quality level of the system.

For this study, defects were examined from different perspectives in order to determine defect distribution pattern and to provide quantitative supporting evidence to the above actions. This study specifically looked at aspects such as who discovered the defects and what major functions had the most defects. In addition, this study also looked at the defects reported by developers verses those reported by testers at different stages of development.

To put the analysis in perspective, a few questions were raised by management. Firstly, what was the impact of enhancing unit tests? Secondly, did introduction of regression testing reduce the rate of defects discovered? In lieu of these questions, this study defined defect count, defect distribution, and reliability growth metrics as a means to analyze and answer the above questions. The defect count metric was defined in order to assist in gauging the magnitude of the defects within entities such

as a component or branch. Defect distribution on the other hand was defined in order to look at the defects across entities such as components or personnel. The reliability grown metric was defined to track the cumulative defects within an entity such as branch over time in order to analyze the reliability growth.

Once the three metrics were defined, this study came up with a main hypothesis as; if in-process feedback is provided earlier in the life cycle of a system, then the purification level ρ will increase closer to one over time. Based on this main hypothesis, the study further defined two more specific hypothesis as; 1, if unit and system tests are enhanced, then more defects will initially be discovered with less defects reported at endpoint. 2, if more unit tests are carried out during development cycle, then the share percentage of defects discovered by developers will increase. These hypothesis will be tested under the ODC validation stage of this research work by looking at the defect count, defect distribution, and reliability growth metrics.

Chapter 4

DEFECT ANALYSIS

This chapter will now go over trend distribution and analysis models [32][47][54], visualizing the whole and sub population of defect distribution. The whole population of defect distribution would be referred to as OED defects while the subpopulation defect distribution would be referred to as branch specific defects.

4.1. Failure Distribution and Data Visualization

The observed failures were recorded on a monthly basis. Consequently, the unfixed defects from previous months were considered in subsequent months. This led to duplicate counting of defects, which conforms to the customer's view of quality before the previously discovered defects were fixed. Figure 4.1 shows a graph on observed failures per month during this study period.

We examined the monthly failure distribution over this entire study period. From Figure 4.1, it can be observed that at the beginning, the failure distribution was relatively low at about 7 observed failures per month. Then the failure instances increased to about 20 per month. This relative high was sustained for about 30 months with fluctuations here and there. The failure occurrences then dropped back down to about 7 failures per month for the rest of the reporting period. Though there were multiple outliers such as the 52 and 36 observed failures in March 2003 and in March 2007 respectively, the data generally showed consistent distribution trend expected of the system.

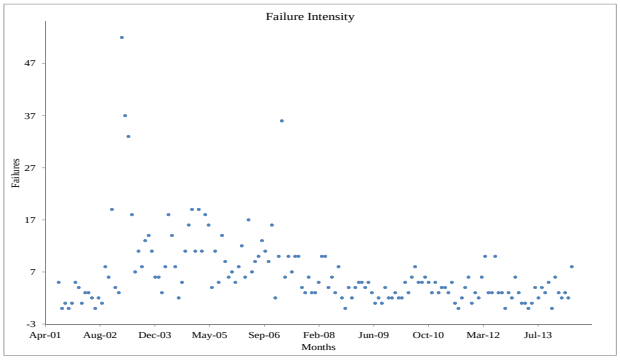

Figure 4.1. OED: Observed failures per month

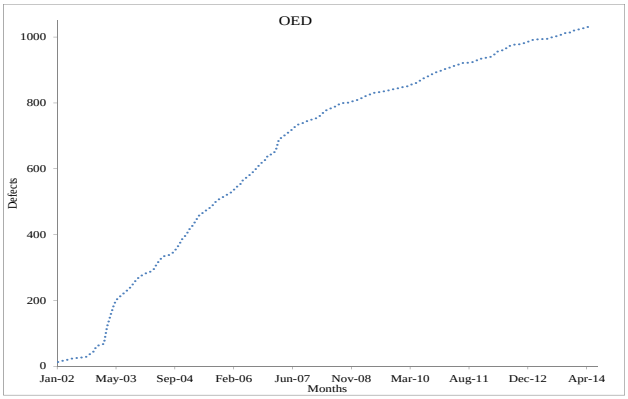

Figure 4.2. OED: Total unique defects over time

4.2. Cumulative Defects Over Time

All the uniquely counted defects were visualized and analyzed by plotting them over the calendar time with the interval in months. The analysis was done by first looking at the whole software entity in Figure 4.2 between August 2001 and June 2014. For a better and more accurate evaluation of the system, we considered the major 4.X, 5.X and 6.X branches in Figure 4.3. The duration of OED, 4.X, 5.X, and 6.X branch was 155 months, 126 months, 93 months, and 34 months respectively.

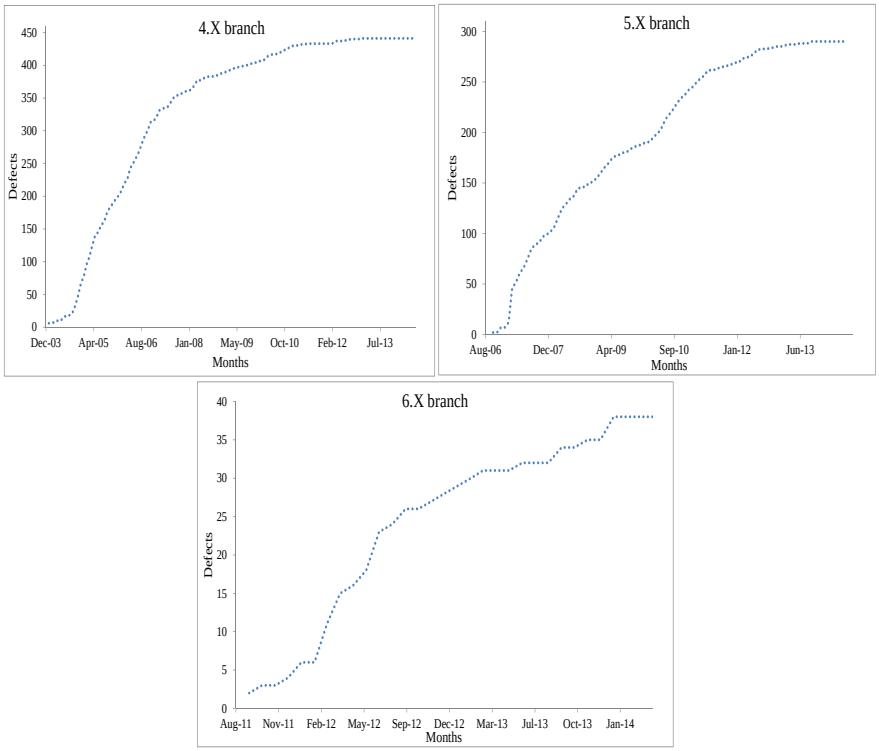

Figure 4.3. Branch: Total unique defects over time

From Figure 4.2 trend, it was observed that the initial defects were relatively flat at the beginning of 2002 with a cumulative defects count of about 50. Thereafter the defect count jumped to about 130 at the beginning of 2003. This was the time the software started to get initial use. End users were now understanding the software and running into defects as they implemented their processes. The defects gradually started to taper off as the months progressed. After a year or so, there was a sudden jump in defects then they tapered off again. The multiple S shape trends coincided with the new feature set being added and deployed. Then newer defects being discovered as the end users went through the learning curve of using the new features and then rolling the system into production.

The 4.X branch trend in Figure 4.3 shows some sort of a concave curve of the defect trend. As the 4.X branch was created around January 2004, there was an immediate ramp up of defects reported. This was a result of most end users who had already worked with the 2.X and 3.X branches during their beta testing adapting the 4.X branch and taking this code base to production. Major emphasis was placed on fixing defects with only necessary and marginal enhancements allowed in the branch. By January 2008, most releases under the 4.X branch were in full production. The 4.X graph shows the trend of defects reported substantially reduced later in the life of the branch as a result of this work. This has been the widely adapted branch with now very limited defects reported as the graph indicates.

The 5.X branch benefited from defects discovered and fixed under 4.X branch. This can be attributed to the newer feature set introduced and end users initially learning how to use the system then discovering newer defects. This happened in several stages of the 5.X branch. Some of the 4.X users migrated to the 5.X branch to take advantage of the new features.

The 6.X branch also benefited from the defects discovered and fixed under the 4.X and 5.X branches. Other than enhancements, the compiler was upgraded to take advantage of newer features and libraries. By the end of this study period, the 6.X branch was not widely adopted in the production environment yet. It was mostly used in the research and development (R&D) environment where users were exploring new features and evaluating production candidate features.

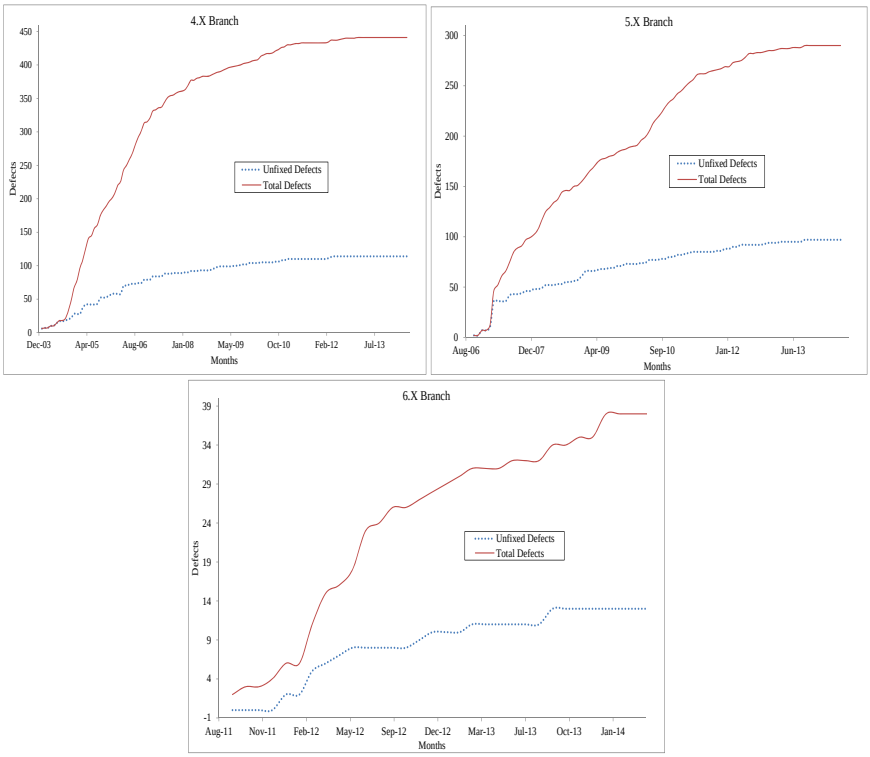

Figure 4.4. Branch: Total and unfixed unique defects over time

4.3. Unfixed Defects Over Time

Naturally, not all reported defects were fixed either due to lower priority or workaround for defects that did not hinder the system's usage. But this data can still be a good indicator of the rate of defects being addressed. Turning attention to the branch specific trend, as seen in Figure 4.4, the unfixed defects trend in the 4X branch seemed to take a concave form indicating more defects were addressed as the months progressed. The rate of addressed defects seems to be higher in 5X branch as compared to the 4X branch.

Looking at the 4X, 5X to 6X branch in Figure 4.4, the number of unfixed defects are relatively low at the beginning of each branch. At this stage the philosophy of addressing defects as soon as possible is now embedded in the culture and new

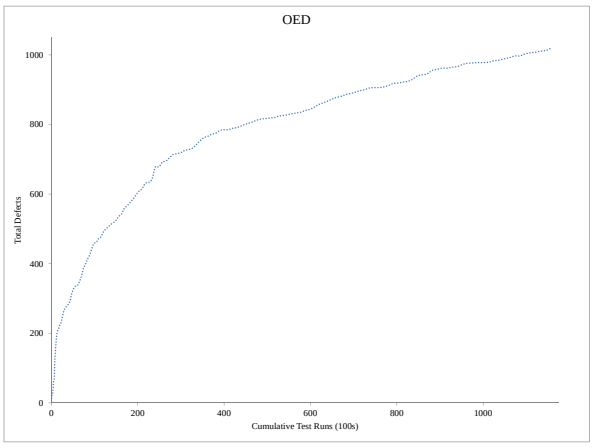

Figure 4.5. OED: Total unique defects over test runs

defects are getting more attention compared to previous releases. This is evident at the beginning of the 5X branch. The unfixed defects ramped up drastically. This was the time most of the effort was placed on the 4X branch and therefore reported defects in 5X were not addressed appropriately. Eventually the 5X branch got its due attention at round March 2007 there onwards. The graph on the 6X branch indicates relatively fewer unfixed defects in a space of 3 years, starting from December 2011 to June 2014. Overall across the board, there was a steady 20% rate of unfixed defects over this study period.

The trend of total uniquely counted defects over cumulative test runs was also considered as another data point of evaluating the reliability of OED. The defects were sorted based on the date they were discovered. The first defects correspond to the defects discovered at the initial point of deployment and subsequent defects were added approaching the nth defect. The nth defect in this scenario corresponds to the latest defect discovered. Similar approach for the runs was applied. The first test runs correspond to the initial test runs with subsequent test runs added cumulatively on a monthly interval. Figure 4.5 and Figure 4.6 indicate the time period the tests

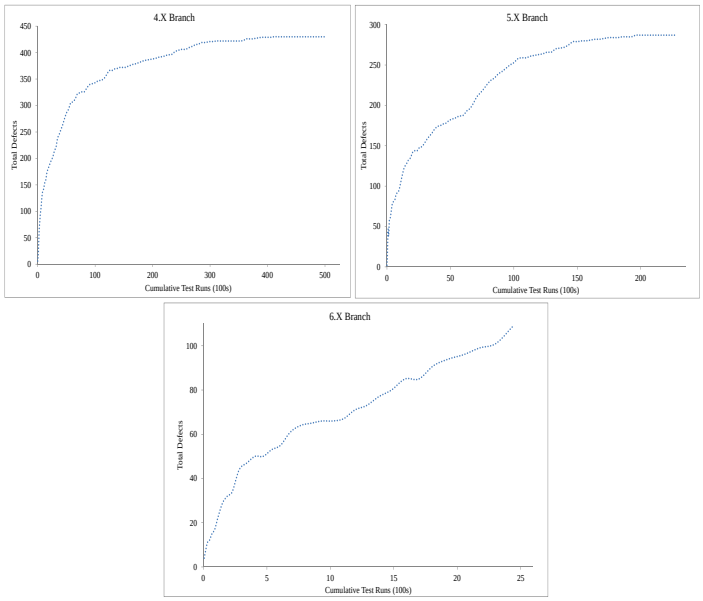

Figure 4.6. Branch: Total unique defects over test runs

were run on a monthly interval over their respective indicated time period.

4.4. Trend Test: Laplace Test on Defects Over Years

The Laplace trend test was applied to the uniquely counted defects for the whole software entity in Figure 4.7 and branch specific trends in Figure 4.8. For both cases, a general trend of improved reliability across the board was observed. The regions labeled as reliability decay and reliability growth corresponded to when the reliability was considered not to be ideal and when the reliability seemed to improve respectively. The region between the reliability decay and reliability growth indicated there was no trend established.

The plots in the reliability decay region, for instance in Figure 4.7, show significant reliability decay peaking at around May 2003 then subsequently turning around and starting to go down as reliability started to improve. The turning point considered

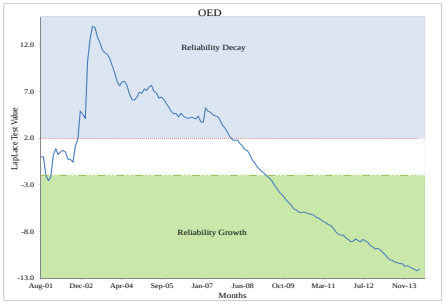

Figure 4.7. OED: Laplace trend test

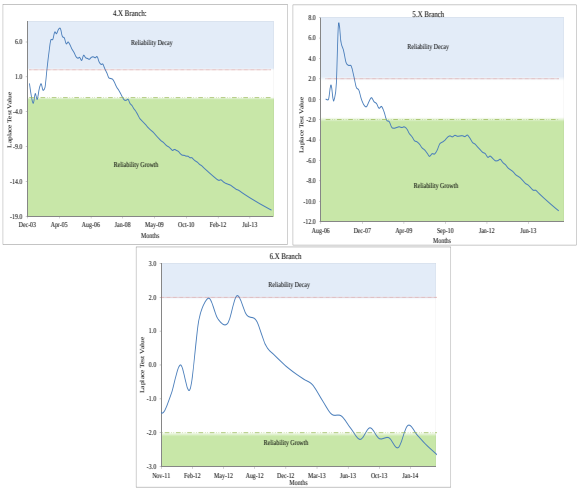

Figure 4.8. Branch: Laplace trend test

as growth in reliability is finally achieved at around April 2009 time frame. But more specifically for the 4.X branch in Figure 4.8, reliability growth is actually achieved around April 2007 and progressively grows till 2014. Release 4.32, which was widely used and deployed to multiple customers, coincides with this date.

For the 5.X branch in Figure 4.8, the turning point for reliability growth per Laplace trend test was around July 2008 which coincided with release 5.2.02. Though relatively speaking, the reliability trend started to reduce at around January 2010,

then peaked at around January 2011, then started to increase till 2014.

The Laplace test for 6.X indicate that the data was not sufficient enough to develop a trend. But looking at the initial plot in Figure 4.8, it does show similarities to the initial stages of the 4.X and 5.X branches before stability. Based on this, we expect that the 6.X branch will eventually exhibit, with time, the reliability growth currently observed in the two mature branches.

4.5. Defects Over Test Runs

Even though Figure 4.2 and Figure 4.3 trends of cumulative defects over calendar month showed a tapering off trend towards the tail end, the overall data and the trend were still not as consistent. This was due to the non-uniform workload in Figure 3.2. Because of this, calender time was deemed less appropriate for reliability growth modeling.

To address the above mentioned concerns, the trend of total uniquely counted defects over cumulative test runs was then considered. The defects were sorted based on the date they were discovered. The first defects correspond to the defects discovered at the initial point of deployment and subsequent defects were added approaching the n^{th} defect. The n^{th} defect in this scenario corresponds to the latest defect discovered. Similar approach for the runs was applied. The first test runs correspond to the initial test runs, with subsequent test runs added cumulatively on a monthly interval. Figure 4.5 and Figure 4.6 indicate the time period the tests were run on a monthly interval over their respective indicated time period.

OED graph in Figure 4.5 shows some reliability growth trend for the whole software entity. The reliability growth trend is more evident in the 4.X branch in Figure 4.6 as the trend seems to take a more concave shape compared to the OED graph in Figure 4.5. The 5.X branch in Figure 4.6, though not as pronounced as the 4.X, does

also exhibit more or less a concave shape.

Even though the 6.X branch also shows some tendency of reliability trend, the shape is not as pronounced as those exhibited by the 4.X and 5.X branches in Figure 4.6. This can partly be explained by the relative immaturity of the 6.X branch as compared to the more mature 4.X and 5.X branches.

We examined Figure 4.5 and Figure 4.6 on unique defects over test runs, then compared the graphs to Figure 4.2 and Figure 4.3 on unique defects over the calendar time. Visually, we observed that the unique defects over test runs gave an overall concave shape more typical of a reliability growth curve.

Chapter 5

Reliability for OED

After an over view assessment of OED's trend distribution and analysis models in Chapter 4, this Chapter will cover the reliability aspects of the OED. The Chapter will look at operational reliability and reliability growth. Under operational reliability, the chapter will look at the IDRM models used to assess the system's reliability, highlighting the stable operational period of the system. Under reliability growth, the Chapter will look at two types of SRGMs, highlighting the long term prediction of the models against the actual raw data.

5.1. Operational Reliability for OED

Operational reliability refers to the system reliability snapshots observed during OED's operation where all the observed failures, including those caused by the same underlying faults yet to be fixed, are counted.

The operation environment for operational reliability evaluation is the actual end users' environment during their normal usage of the system. In addition, failures observed in the lab during the normal testing process after release were also considered. The lab test environment was based on end user's simulated environment. The data used was based on observed failures of the system in the field and internally during normal testing process after release.

Figure 5.1. OED: Success rate

5.1.1. Reliability Assessment using IDRM

With the observation of the expected trend seen in Figure 4.1 above, the next step was to consider the failure rate stability and operational reliability. This was accomplished by assessing the success rate or operational reliability according to Nelson's IDRM for each given month derived by (2.6) during the reporting period. The data assessed in each month was purely on the success rate of the test runs, giving us a series of monthly snapshots of operational reliability.

The success rate plot trend show relative instability at first, indicating the system had a general lower and unstable pass rate. This then transitioned to a much higher passing rate compared to the unstable rate. At this stage, the relatively higher stable rate would indicate the system was stable and fewer defects were being discovered.

Each segment for the data in Figure 5.1 and Figure 5.2 was based on calendar month of the whole software and each branch for the stable reporting period respectively. During each segment, the failed test runs were considered over the total test runs carried out during the segment. The rate was then monitored throughout the stable reporting period for each entity studied. The average success rate or the oper-

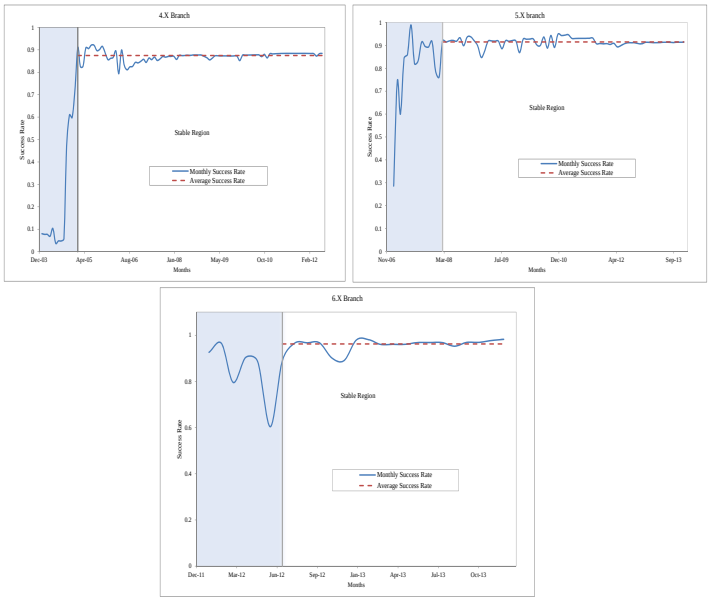

Figure 5.2. Branch: Success rate

ational reliability for the entire stable period according to the Nelson IDRM is shown as the dashed horizontal line in Figure 5.1 and Figure 5.2.

The stable reporting period indicated in Figure 5.1 and Figure 5.2 was derived by the running average process algorithm in Table 5.1, where R_i is the success rate for period i, s is the start of the last stable period, t is the current month evaluated, $X = \{R_s, R_{s+1}, ..., R_t\}$, $Xmax$ is the highest success rate, $Xmin$ is the lowest success rate, and N is the final month of the reporting period. The algorithm determines the stable period from s to N $(i = s, ..., N)$ where every individual success rate R_i falls within a 15% range.

Thus, when algorithm from Table 5.1 was applied to the current data, the stable duration for OED and 4.X branch begun on February 2005, while 5.X branch stable rate begun on March 2008. Looking specifically at OED trend in Figure 5.1, the unstable rate is observed to start at around May 2003 at a rate of 0.08 fluctuating

Table 5.1. Range algorithm

1: **while** t is less than N **do**
2: **if** $(Xmax - Xmin) \geq 0.15$ **then**
3: set s to t
4: **end if**
5: **end while**

to the lowest point of 0.05 in June 2004 then a sudden sharp success rate to 0.90 at around the month of February 2005. This is the time OED was considered to have generally started its stable success period. The 5.X branch started off at a better rate of 0.28 to 0.08 for the 4.X branch in January 2007. Then the rate stabilized to 0.91 in March 2008. This indicated the reliability of both the 4.X and 5.X mature branches was increasing slightly as time passed during the stable period.

The 6.X branch seems to have been the exception to the rule as the first reported period of all the defects were addressed immediately. Though as seen throughout this study, there was inadequate data for the 6.X as compared to the 4.X and 5.X branch to show a trend. Nelson model in this instance does show a high reliability rate of 0.96 which is unusually high compared to the mature 4.X and 5.X branches.

Evaluating the Nelson reliability further, Table 5.2 highlights the average success rates or the operational reliability according to the Nelson IDRM for the entire stable period with its corresponding duration. It gives us the stable and robust operational reliability estimate for OED, 4.X, 5.X, and 6.X branches. The table also shows the total duration of each branch and the Nelson rate at the last branch release month. Looking across the board, the average rate during the stable period and at the last branch release month m was relatively similar with an exception of the 6.X branch.

Table 5.2. Nelson Reliability

		Nelson Reliability			
		Stable Period		At Last Branch Release	
Entity	Total Duration	Rate	Duration	Rate	Month
OED	155	0.877	108	0.889	Jan-14
4.X	126	0.872	90	0.884	Jul-12
5.X	93	0.915	70	0.914	Dec-13
6.X	34	0.957	19	0.982	Jan-14

5.2. Reliability Growth for OED

After assessing the operational reliability above, the next step was to assess the reliability growth of OED. Unlike in operational reliability where duplicate defects were counted, only unique defects were used in reliability growth evaluation. In other words, once a defect was accounted for, it was not counted again in the next test interval. This allowed us to assess the defect fixing effect on reliability growth.

5.2.1. Reliability Growth Assessment using SRGM

The Laplace trend test in Figure 4.7 and Figure 4.8 gave an overview reliability decay and growth trend of OED. Though this trend test gave a good overview of the reliability growth, it did not quantify reliability growth or the amount of reliability improvement over a given period, which is given by SRGMs.

The data used to assess the reliability growth using SRGM was based on the total unique defects over test runs. Unlike with Nelson Reliability graphs in Figure 5.1 and Figure 5.2, the defects in this case were counted only once in order to see the software reliability growth. Though there were incidents of newer defects introduction as a results of defect fixing, from our experience with the tool over the years, these incidents were quite few and as a result, our believe is their impact was quite min-

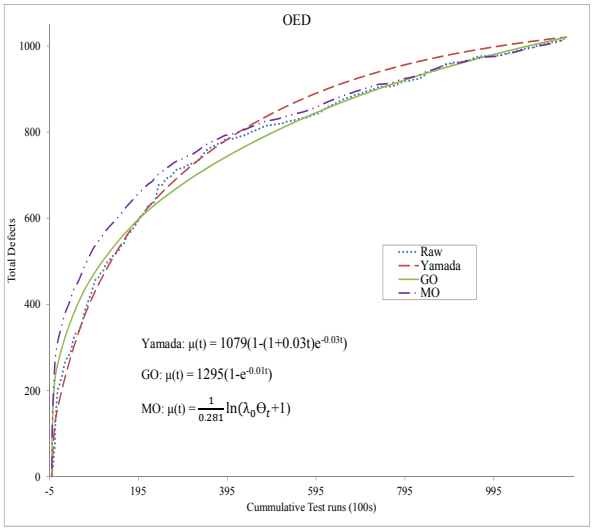

Figure 5.3. OED: SRGM on total unique defects over test runs

imal. Therefore, Yamada, Goel-Okumoto (GO), and Musa-Okumoto (MO) SRGM models[31][82] were fitted to this defect data referred to as raw in all SRGM related graphs.

Figure 5.3 and Figure 5.4 shows OED and branch graphs respectively of the total defects over cumulative test runs trends plotted alongside the fitted Yamada, GO, and MO models. Looking closer at the graphs, it can be observed that the trends follow relatively closely to the raw data for OED, 4.X, and 5.X entities. For the 6.X branch, though the trend was not as close to the raw data as the previous plots, there was still enough evidence to show reliability growth trend.

The goodness-of-fit test was used to determine how well the models fitted. In this case, the R^2 for the three models across the board was very close to 1. For example, goodness-of-fit for Yamada, GO, and MO was 0.996, 0.990, and 0.992 respectively for OED.

In addition to the goodness-of-fit test, purification level ρ define by (2.5) was also used to quantify the reliability growth assessed by Yamada, GO, and MO models by

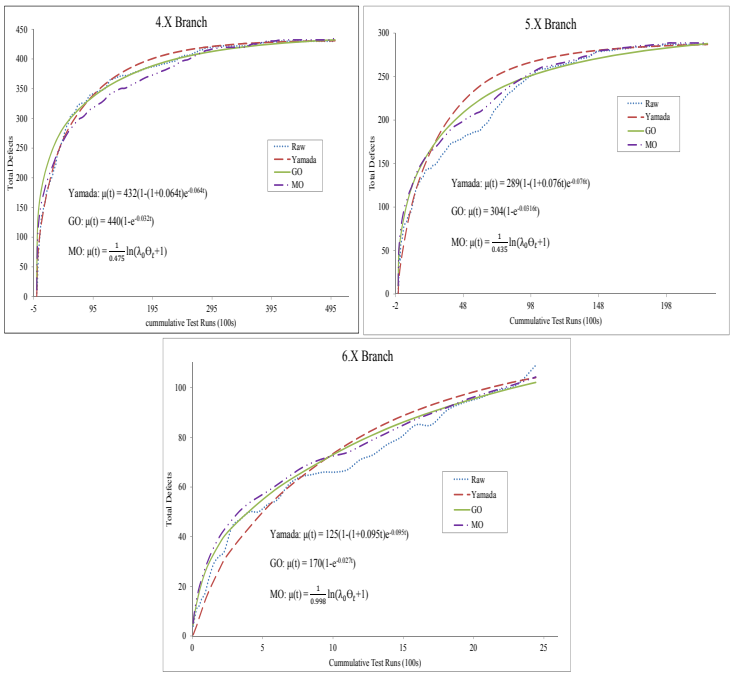

Figure 5.4. Branch: SRGM on total unique defects over test runs

looking at the ratio of the failure rate reduction. Table 5.3 gives the purification level ρ for OED and the branches. Modeling results with SRGMs showed strong evidence of modeling results with reliability growth for the OED, 4.X, and 5.X branches based on the higher purification level ρ. For 6.X branch, the reliability growth was not as strong as 4.X or 5.X and the ρ values estimated from different SRGMs varied considerably. Overall, the high ρ values indicate that continuous defect fixing increased software reliability for this system considerably over time.

5.2.2. Reliability Prediction

Yamada, GO, and MO models were also used for reliability prediction. This was accomplished by dividing the data set into training and testing sets. The training set was determined based on the 50% and 75% training data point of the defects over

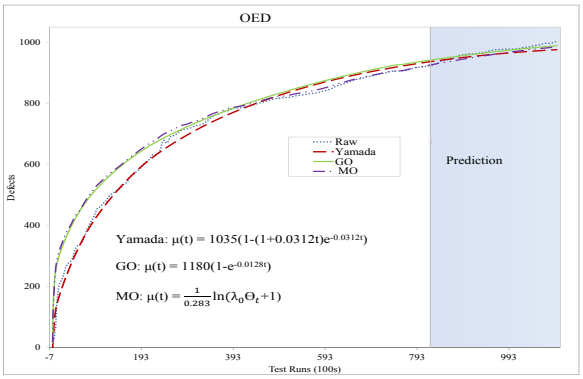

Figure 5.5. OED: Predictions based on 75% Training data

Table 5.3. Purification level ρ

	Entity	Branch		
Model	OED	4X	5X	6X
Yamada	0.9605	0.9933	0.9927	0.7651
GO	0.9988	0.9994	0.9994	0.9581
MO	0.9985	0.9969	0.9997	0.9586

test runs [31][44][57][68]. The 50% and 75% training data set were fitted for all the models to their respective data set. The models were then extended to cover the remaining 50% and 25% periods as testing sets respectively. The fitted and extended data was then plotted next to the actual data from the testing set. Figure 5.5 shows the model predictions for OED and Figure 5.6 shows the predictions model for the 4.X, 5.X, and 6.X branches that used the 75% based prediction data.

Snapshots of the short and long term predictions were examined. The short term prediction was based on approximately one month worth of additional test runs after the cutoff point while the long term was based on the endpoint runs. The predicted failure counts N were obtained by extending the fitted model to the short and long term prediction points and then compared to their respective actual failures. The

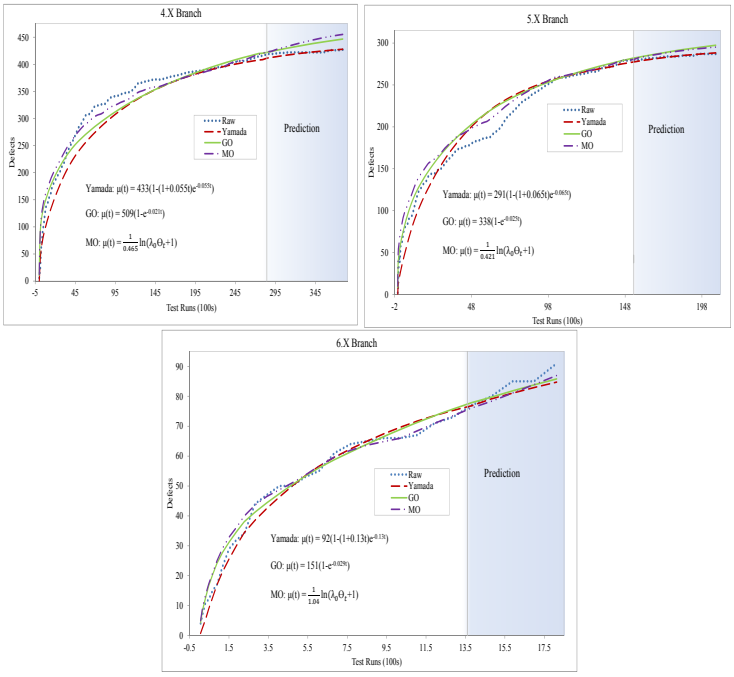

Figure 5.6. Branch: Predictions based on 75% Training data

predicted failure rate λ was derived by taking the slope of defects over total test runs for each type of prediction model. The actual failure rates were based on the first and last month for the short and long term prediction points.

The prediction results for the 50% and 75% training sets are shown in Table 5.4 and Table 5.5 respectively, highlighting the cell with a model that was closer to actual data. Looking at OED's 50% prediction based data, it can be observed that GO predicted closer to reality for long term case, while MO predicted better for short term case. For the 75% prediction based data, MO seemed to have prediction closer to reality for both short and long term cases. For branch specific data, the mature 4.X and 5.X branches, Yamada seemed to predict better in short and long term for both prediction based types. While MO seemed to predict better for the less mature 6.X branch for both short and long terms. Overall all the models predicted fairly

Table 5.4. Prediction Based on 50% Training Data

		Short Term (Month)		Long Term (At Release)	
Entity	Prediction Type	Failure(N)	Failure Rate (λ)	Failure(N)	Failure Rate (λ)
OED	Actual	835.00	0.00197	1003.00	0.00491
	Yamada	849.37	0.00333	957.75	0.00127
	GO	868.99	0.00414	1025.70	0.00229
	MO	839.59	0.00262	939.26	0.00179
4.X	Actual	388.00	0.00207	427.00	0.00189
	Yamada	380.21	0.00321	420.38	0.00165
	GO	389.48	0.00331	431.17	0.00173
	MO	383.31	0.00325	424.26	0.00169
5.X	Actual	259.00	0.00309	287.00	0.00111
	Yamada	256.77	0.00527	285.00	0.00118
	GO	258.75	0.00729	306.36	0.00301
	MO	255.33	0.00779	313.80	0.00458
6.X	Actual	66.00	0.00591	91.00	0.05797
	Yamada	67.66	0.01992	79.07	0.01839
	GO	68.41	0.02318	82.32	0.02166
	MO	68.58	0.01988	79.88	0.01834

Table 5.5. Prediction Based on 75% Training Data

		Short Term (Month)		Long Term (At Release)	
Entity	Prediction Type	Failure(N)	Failure Rate (λ)	Failure(N)	Failure Rate (λ)
OED	Actual	940.00	0.00877	1003.00	0.00491
	Yamada	941.87	0.00201	975.94	0.00086
	GO	947.44	0.00214	988.94	0.00129
	MO	933.08	0.00269	984.69	0.00132
4.X	Actual	421.00	0.00192	427.00	0.00189
	Yamada	415.14	0.00223	427.72	0.00111
	GO	427.12	0.00301	446.61	0.00207
	MO	429.18	0.00425	455.21	0.00218
5.X	Actual	281.00	0.00285	287.00	0.00111
	Yamada	279.51	0.00279	288.24	0.00118
	GO	284.40	0.00358	297.33	0.00220
	MO	283.91	0.00351	295.16	0.00154
6.X	Actual	80.00	0.03000	91.00	0.05797
	Yamada	79.16	0.01889	84.74	0.01783
	GO	79.72	0.02050	85.82	0.01944
	MO	78.34	0.02540	86.93	0.02574

Table 5.6. Correlation factor based on raw and prediction data

		Entity			
Model	Training Data	OED	4.X	5.X	6.X
Yamada	50%	0.9936	0.9398	0.9658	0.9790
	75%	0.9734	0.7232	0.9632	0.8976
GO	50%	0.9942	0.9406	0.9770	0.9790
	75%	0.9760	0.6952	0.9632	0.8974
MO	50%	0.9909	0.9399	0.9769	0.9837
	75%	0.9722	0.6983	0.9581	0.9002

well.

Table 5.6 highlights the R^2 correlation factor between the prediction and actual data for all the models and prediction type based on the 50% and 75% based training data set. From this data, the general distribution R^2 across the board was closer to 1, indicating a good fit. The exception was with the 75% based training data set prediction type for the 4.X branch for all the models where R^2 was less than 0.8.

Chapter 6

Orthogonal Defect Classification (ODC)

Previous chapters looked at the trend distribution and analysis models used in this study for visualizing the whole and sub population of OED defect distribution. In addition, operational reliability and reliability growth was discussed in more details particularly using IDRM and SRGMs models. To bridge the gap between the defect and reliability analysis discussed previously, this chapter will go over the scaling down and adaption of the original ODC attributes that were relevant for the OED system to provide in-process feedback for focused quality and reliability improvement.

The magnitude of defects reported in the early years of the system's life cycle caused concerns to stakeholders. Consequently, ODC was adapted to analyze OED defect data and multiple policies were implemented to address these concerns. This study on ODC analysis looked at defect data before and after ODC to understand the impact of these policies.

6.1. Adapting ODC for OED

Historically, original ODC from IBM study[21] has been adapted in other studies to meet their needs [27][53][56][6]. Similarly, this study revisited the standard ODC attributes by scaling down and adapting them to OED system analysis needs. Table 6.1 highlights the ODC attributes that were meaningful to the OED system. Just like the original ODC, this study categorized the attributes under opener and closer sections. The column labeled original under the attribute main column indicates what attributes were adapted from the original ODC attribute.

Table 6.1. Orthogonal Defect Classification Attributes

	Attribute		Attribute Value
Section	Original	OED	OED
Opener	Defect Removal Activity	Activity	Unit Test
			System Test
			Customer Usage
			Other Activities
	Triggers	Trigger	Logic
			Hardware
			Backward Compatibility
			Other Triggers
	Severity	Severity	Production Stop
			Average
			Minor Severities
	Discovered By	Discovered By	End User
			Developer
			Tester
			Other Personnel
Closer	Defect Type	Defect Type	Algorithm
			Communication
			Processing
			Other Functions
	Source	Branch	4.X branch
			5.X branch
			6.X branch
			Other branches
	Source	Component	Processor
			Computational Component
			Sensor Communicator
			Tool Communicator
			RF Communicator

Looking closer at the attributes for OED, under the opener section, this study defined the *Activity, Trigger, Severity*, and *Discovered By* attributes. This study scaled down the original ODC attributes *Defect Removal Activity*, *Triggers*, and *Severity* and adapted them as *Activity*, *Trigger*, and *Severity* attribute respectively for OED.

The original *Defect Removal Activity* attribute covered defects found during activities such as design review, code inspection, and different types of tests. The OED *Activity* attribute was scaled down and adapted to *Unit Test* and *System Test* attribute values to track defects discovered during the unit and system test activity respectively. In addition, the study adapted *Customer Usage* attribute value to track defects reported based on the customer's usage and *Other Activities* attribute value to track other defects based on usage not accounted for by the already defined attributes.

For the OED *Trigger* attribute, the original *Triggers* attribute cover defects from design review and different test triggers. This study then scaled it down and to cover specific triggers from unit and system test only, adapted *Logic, Hardware, Backward Compatibility,* and *Other Triggers* attribute value to track defects based on any logic in the source code, hardware causes, backward compatibility related and other causes not falling in the defined categories respectively.

The original ODC *Severity* attribute measures the degree of the impact a defect has. For the OED system, the study adapted the original *Severity* attribute to look at the level of severity the defects impacted the end user. Consequently, this study defined *Production Stop* and *Average* attribute values to track the major and average level of impactful defects respectively. The remainder of the defects were tracked under the *Minor Severities* attribute values.

The *Discovered By* original attribute tracked the personnel that discovered the defects. Since this attribute was applicable to OED, this study adopted the original attribute as is and defined OED specific attribute values as *End User, Developer,*

Tester, and *Other Personnel* to track defects reported by them respectively.

Similarly in the closer section, *Defect Type*, *Branch*, and *Component* attributes were defined for this study based on both the needs and the availability of reliable data. The attribute *Defect Type* was adopted directly from the original ODC that represented the actual correction. The *Branch* and *Component* attributes were adapted for OED from the original ODC *Source* attribute. The original *Source* attribute define where the defects are found that come from either in-house or other sources.

For the *Defect Type* attribute, the *Algorithm*, *Communication*, *Processing*, and *Other Functions* attribute values were defined to track defects related to proprietary algorithms, protocol and general communication, general data processing, and other minor functions not already categorized respectively. In respect to *Branch* attribute, the attribute values *4.X branch, 5.X branch,* and *6.X branch* were defined to highlight the defects based on the 4.X, 5.X, and 6.X released branches respectively. The other branches this study tracked using *Other Branches* attribute value.

This study defined for the *Component* attribute, the attribute values *Processor, Computational Component, Sensor Communicator, Tool Communicator*, and *RF Communicator* to track and highlight defects for a set of modules grouped together to form an executable or dynamic linked library for Processor, Computational Component, Sensor Communicator, Tool Communicator, and RF Communicator respectively.

The remainder of the following omitted attributes; *target, qualifier*, and *age* under the closer section were deemed as not applicable to this study due to the focus of dealing with design, missing implementation, and not reliable data available for the *target, qualifier*, and *age* attributes respectively. Therefore, no further consideration was placed on these attributes.

6.2. ODC Data Analysis and Validation

Once this study defined the ODC attributes, an ODC scheme was defined to analyze the different aspects of the defect trend data to understand the impact of the policies adopted to the system. The ODC analysis process was defined in three stages. First step involved an analysis of the whole defect distribution of the system between the initial release and the time the above policies were implemented. ODC baseline was defined as prior to 2006, which included data reported up to December 2005. The defect count, defect distribution, and reliability growth baseline metrics was then established to be used for comparison in the third step.

The second step involved an iterative ODC analysis process where the first iteration looked at individual ODC attribute. For this analysis, this study looked at the defect distribution of the major functions and where the defects were discovered. Then based on the result of the first iteration, if additional information was needed, subsequent iteration were looked at as a combination of the ODC attributes to fill in the gaps highlighted in the prior iterations.

The third step looked at the distribution of defects for the remainder of the reporting period. Objective was to compare the end of reporting period to the baseline reporting period and see the effect of the changed policies and determine the viability of bridging the gap between defect casual and statistical analysis. This was achieved by comparing post ODC to baseline ODC defect count, defect distribution, and reliability growth metrics.

6.3. Baseline Analysis

This section will go over the ODC analysis, beginning with defect distribution of the baseline, followed by an ODC analysis in an iterative way through the attributes identified in Table 6.1. First, this study examines *Defect Type* and *Discovered By*, two attributes that showed the most interesting patterns among all the attributes this study examined. Then, the study analyzed the interaction between *Defect Type* and *Branch* attributes.

6.3.1. Defect Type Attribute

Objective of using the *Defect Type* attribute was to observe the defect distribution across the major functions that included algorithm, communication, and processing. The key question for this analysis was to determine what function was more defect prone and whether this was within expectation. Based on the analysis, a reasonable explanation would be expected to be reached on the defect distribution.

Figure 6.1 highlights defects for the three major functions of OED defined as the baseline defined in the first stage. From Figure 6.1, it can be noted that the processing related defects were the most reported. This was not surprising as the share amount of data processed was high and varied across the board, leading to a higher probability of failures.

The next most reported defects were communication related. Initially this was a surprise as the algorithm defects were expected to be more compared to communication defects. On further analysis, it was realized that the algorithms were initially developed on a separate platform and went through necessary validation process. Thus, the fundamental principles were considered to be solid. Which led us to conclude the defects reported were more likely related to the integration process to a new environment.

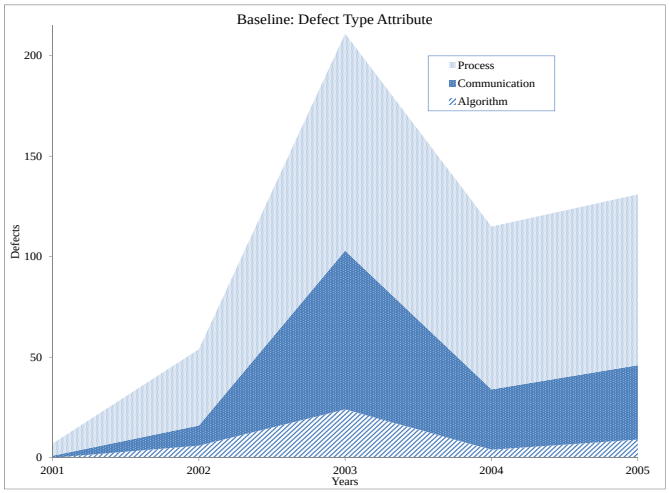

Figure 6.1. Baseline: Defect Type Attributes Trend

6.3.2. Discovered By Attribute

From the *Defect Type* attribute analysis showing an increased defect reported, this study needed to understand at what stage of the software cycle the defects were being discovered. For this analysis, this study looked at *Discovered By* attribute where defects were discovered, either in-house or in-field. This addressed the one of the major concerns for management of limiting unknown defects being discovered first by the end-user.

Figure 6.2 highlights the baseline defects discovered by testers, developers, and customers during the normal system test, unit test, and in-field use respectively. At first glance, defects reported by the end-user were observed to be less compared to those discovered in-house. In addition, the defects reported by the end-user were observed to be relatively consistent across the years. This at minimum settled management's concern of less defects being reported from the field compared to those from in-house testing.

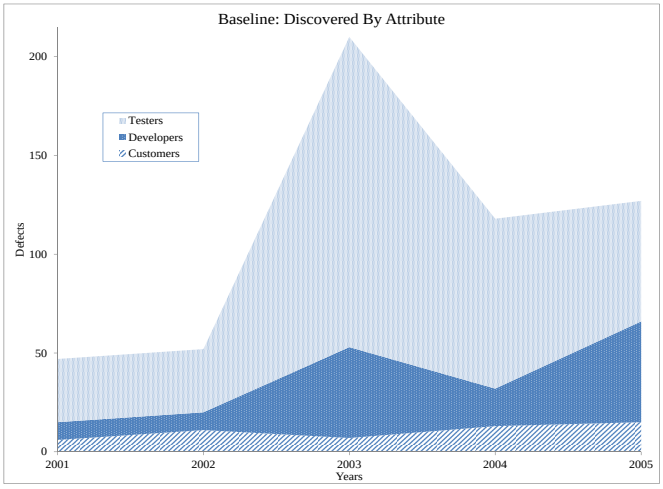

Figure 6.2. Baseline: Discovered By Attributes Trend

Table 6.2 shows the baseline ratio of defects discovered by each personnel over the total number of defects for each year, highlighting the higher reported defect between the three reporting entities. Looking at the testers' and developers' reporting rate in Table 6.2, it was observed that the testers had significantly higher reporting defects between 2001 and 2004 reporting period. The gap closed in 2005 though the testers rate remained higher. This data supported a known fact of developers not executing heavily unit tests before delivering code to the testers.

6.3.3. Defect Type and Branch Attributes

To put the defect trend in Figure 6.1 in perspective, this study took a second iteration analysis by looking at the *defect type* and *Branch* attribute interaction. There was interest in looking at how the major functions' defect trends faired on between branches. Since the 4.X branch was the latest branch between 2001 and 2005 time period, a decision was made to look back to the 3.X branch and observe the trend between the 3.X and 4.X branch. Table 6.3 highlights the peak of the

Table 6.2. Baseline: Defect reporting rate

	In-House			In-Field
Year	Developer	Tester	Total	Customer
2001	19%	68%	87%	13%
2002	17%	62%	79%	21%
2003	22%	75%	97%	3%
2004	16%	73%	89%	11%
2005	40%	48%	88%	12%
Average	23%	68%	88%	12%

Table 6.3. Baseline Defect distribution across functions and branch

	Branch	
Function	3.X	4.X
Algorithm	8	8
Communication	39	40
Processing	44	129
OED	91	177

defects of the major functions across the 3.X and 4.X branches. The peak period contained the total defects reported two years prior to the year the most defects were reported.

From Table 6.3, it was observed that between the 3.X and 4.X branches, the Algorithm and Communication functions defects were about the same across the branches. Though, the processing function showed a three times increase in defect between the 3.X and 4.X branches.

From the above analysis, the general trend of defects observed were increasing, particularly under the processing function and not showing signs of declining. In addition, most of the defects were being reported by the testers. Naturally this was a concern for the stake holders. Consequently, a decision was made requiring unit test

to be consistently be performed in order to ease the workload of the testers and allow defects to be discovered earlier in the process. In return, the testers could spend more time on regression testing.

6.4. Validation

After the above analysis, the next appropriate step was taken to show the effect of the changed policies. This section will address the validation process by comparing the defect count across the functions and branches, defect and workload distribution across the personnel involved, and reliability growth of the system between baseline and post ODC. The attributes to be used for this analysis will include *Defect Type*, *Branch*, and *Discovered By* attributes.

6.4.1. Defect Count

To analyze the attributes further, Table 6.4 highlights the total defects under the 4.X and 5.X, showing the share percentage across the functions within the branches. Across the board, the 5.X branch defects reported were fewer than those reported in the 4.X branch, underscoring the system's overall quality improvement across the newer branch. In addition, the Algorithm defects reported remained at a steady rate of 11% while communication defects increased by 10% while the processing defects reduced by 9% share. Processing continued to be the most reported defects among the functions.

6.4.2. Defect Distribution by Personnel

One of the challenges in a daily development environment is enforcing certain policies without impacting very aggressive schedules. To put the trends in Figure 6.2 on the ODC baseline and Figure 6.3 on post ODC in perspective, this study took the ratio of defects discovered by each personnel over the total number of defects for each

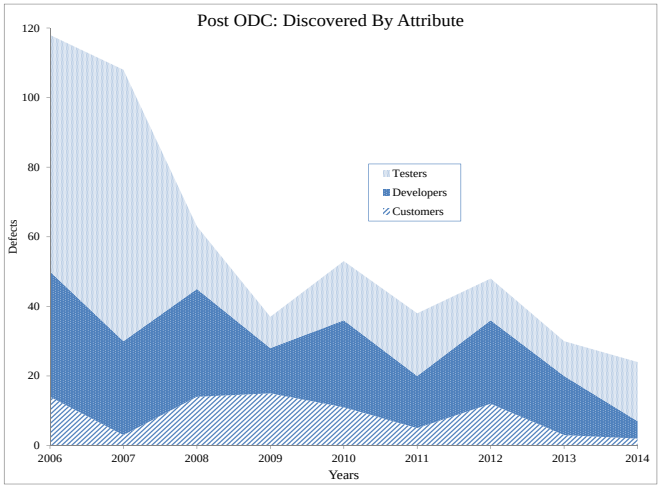

Figure 6.3. OED: Post ODC Discovered By Attributes Trend

Table 6.4. Total Defects within the Branches

Functions	4X		5X	
	Defects	Share percent	Defect	Share percent
Algorithm	43	11%	29	11%
Communication	103	25%	96	35%
Processing	259	64%	151	55%
OED	405	100%	276	100%

year, then observed the trend of the defects ratio over the reporting period.

Table 6.5 shows the trend of defect ratio described above for each personnel discovered defects in post ODC, highlighting the higher reported defect between the three reporting entities. Table 6.5 looks at the effect of the policy change of developers carrying out more unit tests before transitioning to formal testing. Generally speaking, it was observed that the rate of reporting for the developers on average increased after the policy change.

Table 6.5. Post ODC: Defect reporting rate

	In-House			In-Field
Year	Developer	Tester	Total	Customer
2006	31%	58%	88%	12%
2007	25%	72%	97%	3%
2008	49%	29%	78%	22%
2009	35%	24%	59%	41%
2010	47%	32%	79%	21%
2011	39%	47%	87%	13%
2012	50%	25%	75%	25%
2013	57%	33%	90%	10%
2014	21%	71%	92%	8%
Average	39%	44%	83%	17%

Even though defect reporting was higher for testers over the developers in the first two years, there were about four years where the developer's had a higher reporting rate and four years where the testers had a higher rate, with an outlier of the customer reporting higher than the other two in-house personnel in one year. The four years the developers had a higher reporting rate was in 2008, 2010, 2012, and 2013. While the testers' higher reporting rate occurred in 2006, 2007, 2011, and 2014. The one year ourlier of where the customer reported higher rate compared to the other two personnel was in 2009. Despite this, the reporting rate was still lower compared to the other two personnel combined, indicated as Total under In-House column in the table.

At a high level observation from both Table 6.2 and Table 6.5, this study looked at the average reporting rate for each respective reporting period. Looking at the developer's reporting rate between baseline ODC and post ODC, it was observed that the rate increased from 23% to 39%. On the other hand, the testers rate reduced from 68% to 44% with the customer rate rising slightly from 12% to 17%. Though this

Table 6.6. Personnel based defect discovered distribution rate

Personnel	Baseline		Endpoint		Share Distribution Change
	Defect	Share Percent	Defect	Share Percent	
Customer	20	6%	5	9%	3%
Developer	65	20%	22	41%	21%
Tester	243	74%	27	50%	-24%
OED	328	100%	54	100%	

study does acknowledge other facts could also have contributed to this trend, but the study can also argue the more unit test activities were carried out as more defects were being discovered earlier in the process of development before formal test. This supports the hypothesis; if more unit tests are carried out during development cycle, then the share percentage of defects discovered by developers will increase.

Looking at the customer defect reporting rate, it was observed that the defect ratio remained relatively steady across time at around 20% in the post ODC time period. On one hand, the defects discovered by the end user were still much fewer compared to those discovered in-house. On the other hand, the limited knowledge of the customer's operational profile was still an issue and did not change the status quo.

Table 6.6 further highlights the defect distribution across the personnel between the baseline and endpoint. From the table, it was observed that the workload was shifted from the testers based on the 24% reduction at endpoint. On the other hand, the developer share percentage increased by 21%. This results further shows evidence the policy did take effect as was hoped. The customer's share distribution rate increased slightly by 3%, support the limited propriety knowledge was still playing a small role in the field reported defects.

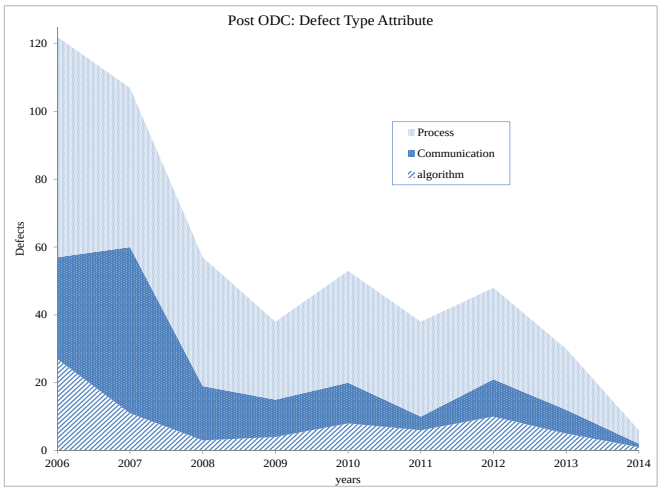

Figure 6.4. Post ODC: Defect Type Trend

Table 6.7. Defect Count Across Functions and Branches

	Peaks		EndPoint	
Branch	4.X	5.X	4.X	5.X
Algorithm	8	11	6	8
Communication	40	57	7	2
Processing	129	75	12	11
OED	177	143	25	21

6.4.3. Defect Distribution Over Time

From Figure 6.4, it was observed that defects after ODC was applied were seen to have reduced as time progressed. To analyze the trend of defects in Figure 6.4, this study looked at the interaction between the *Defect Type* and *Branch* attributes over the duration of the study period represented in Table 6.7.

Table 6.7 highlights the defects across the functions and the whole system for each branch, comparing the defects across the branches and from the peak of the defects reported and at the end of the reporting period, endpoint. The peak period contained the total defects reported two years prior to the year the most defects were reported.

While the endpoint period indicated the total defects reported two years prior to the end of the reporting period.

Looking at the peak and endpoint period across the whole entity in Table 6.7, the defects were substantially reduced. This also indicated the system was exhibiting some quality improvement. Though from the peak, processing functions had a substantially high number of defects reported, by the end of the reporting period, all the functions had relatively similar number of defects. These results support the hypothesis that states; if unit and system tests are enhanced, then more defects will initially be discovered with less defects reported at endpoint.

6.4.4. Reliability Growth

Since ODC baseline occurred right around 2006, 4.X branch partially ended up falling in the ODC baseline time frame. For validation, the defects report during the ODC 4.X branch was evaluated and compared to the 5.X branch that was created after ODC implementation time frame. The time frame for comparison in the 5.X branch was equivalent to the time frame the 4.X branch was created to the end of the ODC baseline. This translated to about 24 months or about 2230 test runs after creation of the 4.X branch.

For comparison of ODC baseline and post-ODC branches, this study took the 4.X and 5.X data collected within the 24 months and fitted the raw data with SRGMs of their respective trend. Purification level ρ for each model was then determined and compared across the branches. Figure 6.5 highlights three graphs. The first graph shows the 4.X and 5.X branch reported defects across the 2230 test runs. The second and third graphs show the 4.X and 5.X branches fitted with the GO and MO models respectively. Based on these fitted models, the purification level ρ were compared side by side in Table 6.8.

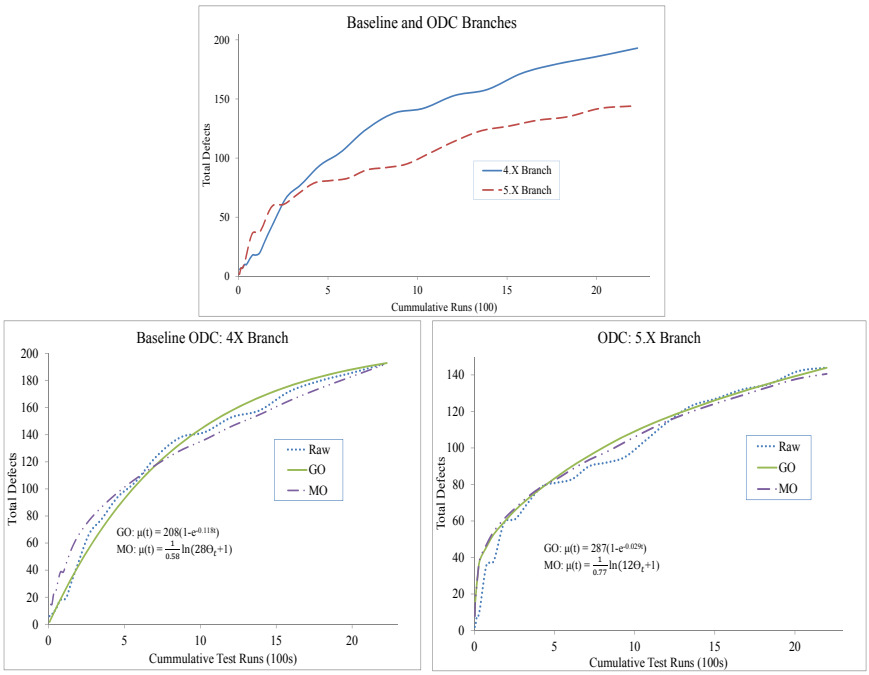

Figure 6.5. ODC Baseline 4X and 5X ODC

Table 6.8. Purification Table for OED over Baseline and ODC based Branches

Model	4.X Branch	5.X Branch
GO	0.916	0.991
MO	0.902	0.994

From Table 6.8, it can be seen that the overall purification level ρ is significantly higher in the 5.X branch across the two models. This subsequently supports the main hypothesis, if in-process feedback is provided earlier in the life cycle of a system, then the purification level ρ will increase, indicating significantly more reliability growth.

6.5. ODC Summary and Conclusion

To summarize, this study looked at the defects in three ways. First this study looked at general trend of the major functions by using the *Defect Type* attribute. This analysis highlighted Process function as being the most defect prone. To break it down further, this study looked at these functions across the branches via a two way analysis using the *Defect Type* and *Branch* attributes. The two way analysis showed clearly the Process function had about three times more defects between the 3.X and 4.X branches.

This study was instrumental in bridging the gap between causal and statistical defect models using ODC in a small company. Similar to other studies on ODC [53][6], this study adapted original ODC attributes [21] to meet this study's need under the specific environment constraints. This study developed an ODC scheme to assist in analyzing defect distribution where they were discovered and the major functions.

To assess the impact of this in-process feedback, this study first quantified the baseline defect count, distribution, and reliability growth. Initial analysis based on where the defects were discovered showed a need to enhance unit tests and additional regressions test suite. Subsequently, after adoption of these policies, this study compared the results and demonstrated that ODC offered valuable early in-process feedback that led to quantifiable improvement using the same metrics.

After the baseline date, this study did show trend of defects decreasing. Using defect reporting rate by personnel, this study showed the testers with a higher reporting rate at the baseline did eventually show a reduced reporting rate of about twenty percent post ODC. This study also showed the developers rate increase by about twenty percent from the baseline time frame. The validation process showed that the policy did not impact negatively in defect detection in-house but ensured effort distribution was retained within the in-house personnel.

Additional validation process showed across two branches, one branch created prior to ODC and the other branch created post ODC. The branches were compared quantitatively using purification level ρ and showed the branch created post ODC did show better results. This was also backed up by looking at the total defects between the branches. The post ODC branch did show fewer defects reported compared to the earlier branches. This validated the positive impact of the policies through reliability growth of the system.

The above analysis showed evidence of most defects being caught in-house before deployment. From the results, this study concluded management expectation of limiting defects discovered first in-field was mostly achieved. From experience, the relatively small discovered defects in the field could partially be explained as those missed due to limited lack of knowledge of the proprietary operation profile of the end-users.

Overall, this study has showed practical ability of using ODC with limited data found in small companies to provide valuable in-process feedback to developers, for the purpose of improving software quality and reliability.

Chapter 7

SUMMARY and CONCLUSIONS

This chapter will look at the summary and conclusions of this study.

7.1. Summary

The main objective of this study was to provide OED stakeholders evidence on it's reliability across multiple releases. The challenge for the study was accessing typical data normally available to large companies that is required by SRGMs and IDRMs to assess reliability [55][64][78]. Thus, there was a need to determine whether other types of data normally available to small software development organization could also be suitable to these techniques. The first step was to account for the reality of working in an ever changing, unpredictable environment with limited resources and data availability, that has a diverse, unique, and customized end user operation that could be very limiting in analyzing the software system. Despite this, successful analysis of the system was achieved by analyzing the defect data over time and system usage using Laplace trend test, IDRMs, and SRGMs.

Building upon previous work, the Laplace trend test was used to demonstrate an overall reliability growth trend presented in the following conference paper:

- "Defect Analysis over Multiple Release Versions of a Semiconductor Software System" paper present at the IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems(COUFLESS) workshop in 2015.

In addition, the observed failures were used in IDRMs to demonstrate when the system reached stability and to obtain a robust reliability estimate for the extended stable period. The unique defects over test runs were used in SRGMs to evaluate the reliability growth of OED. The accuracy demonstrated by such modeling results gave credibility to any forecast applied to mature branches as resented in:

- "Reliability over consecutive releases of a semiconductor optical endpoint detection software system developed in a small company" published in The Journal of Systems and Software, vol. 137, pp. 355-365, in March 2018.

Since in-process feedback is essential in providing useful information to stakeholders decisions of improving software quality and reliability, previous work also looked at adapting the original ODC for OED use. The study demonstrated the ability of providing in-process feedback using data that is normally found in a small company. This was accomplished by first quantifying the baseline defect count, distribution, and reliability growth as presented in:

- "Defect classification and analysis in a small company" paper presented at the 31st International Conference on Computer Applications in Industry and Engineering (CAINE 2018) conference in October 2018.

In following up with the original ODC adaption for the OED system paper [2], the next paper demonstrates how to quantify quality improvement using the defect rate, distribution, and reliability growth metrics after actions that resulted from the in-process feedback. The comparison results demonstrates that ODC offered valuable early in-process feedback that led to quantifiable improvement. The plan is to submit this paper to one of the major journals for review and publication.

7.2. Conclusions and Perspective

Determining software reliability across multiple software releases is essential to end users. This helps them make informed decisions on upgrading software releases without impacting significantly their characterized processes and software quality standards. This was achieved by analyzing the standard defect data normally collected in a typically small software development organization.

For the initial analysis of the data, Laplace trend test results was used to give a quick assessment and visualization of the reliability growth trend. Then IDRMs operational reliability analysis results was used to provide a stable and robust reliability estimates for an extended stable period after some initial fluctuation. Finally, the reliability growth analysis results of SRGMs provided evidence that continuous defect fixes increased software reliability substantially over time.

In concluding this study, ODC analysis was achieved by successfully scaling down and adapting the original ODC attributes. This study demonstrated how to quantify quality improvement using the defect count, distribution, and reliability growth metrics after actions were taken from the in-process feedback. The comparison results demonstrated that ODC offered valuable early in-process feedback that led to quantifiable improvement.

Long term plans are to adopt techniques used in this study to similar environment and to adapt it to other industries with less proprietary concerns where operational data can be accessed more easily. With this additional data, more metrics can be added to the analysis with an expectation to have a much better reliability growth forecasting for live and relatively new branches.

Even though many challenges were faced, multiple measures were taken such as developing meticulous process of categorizing failures to overcome these challenges. In conclusion, this study has demonstrated effective reliability assessment of a software

system within a limiting environment that can be very beneficial to anymore that has access to defect data with limited resources.

REFERENCES

[1] Abuta, E., and Tian, J. Defect analysis over multiple release versions of a semiconductor software system. *IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems(COUFLESS)* (2015), 55–61.

[2] Abuta, E., and Tian, J. Defect classification and analysis in a small company. *31st International Conference on Computer Applications in Industry and Engineering (CAINE 2018)* (October 2018), 175–182.

[3] Abuta, E., and Tian, J. Reliability over consecutive releases of a semiconductor optical endpoint detection software system developed in a small company. *The Journal of Systems and Software 137* (March 2018), 355–365.

[4] Agrawal, M., and Chari, K. Software effort, quality, and cycle time: A study of cmm level 5 projects. *IEEE Trans. on Software Engineering 33*, 3 (Mar. 2007), 145–156.

[5] Alaeddine, N., and Tian, J. Analytic model for web anomalies classification. In *Proc. 10th IEEE Int. Symp. on High Assurance Systems Engineering* (Nov. 2007), pp. 395–396.

[6] Alannsary, M., and Tian, J. Cloud-ODC: Defect classification and analysis for the cloud. *International Conference on Software Engineering Research and Practice (SERP'15)* (July 2015), 71–77.

[7] ANSI/AIAA. *Recommended Practice for Software Reliability.* No. ANSI/AIAA R-013-1992. American Institute of Aeronautics and Astronautics, Feb. 1992.

[8] Asad, A., Ullah, M. I., and Rehman, M. J.-U. An approach for software reliability model selection. *IEEE Proceedings of the 28th Annual International Computer Software and Applications Conference 1* (2004), 534–539.

[9] Basili, V. Quantitative evaluation of software methodology. *Proceedings of the First Pan Pacific Computer Conference 1* (September 1985), 379–398.

[10] Basili, V. R., and Rombach, H. D. The TAME project: Towards improvement-oriented software environments. *IEEE Trans. on Software Engineering 14*, 6 (June 1988), 758–773.

[11] Bassin, K., and Santhanam, P. Managing the maintenance of ported, outsourced, and legacy software via orthogonal defect classification. *Proceedings IEEE International Conference on Software Maintenance. ICSM 2001* (November 2001), 726–734.

[12] Bassin, K. A., Kratschmer, T., and Santhanam, P. Evaluating software development objectively. *IEEE Software 15*, 6 (Nov./Dec. 1998), 66–74.

[13] Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., and Chillarege, R. A case study of software process improvement during development. *IEEE Trans. on Software Engineering 19*, 12 (Dec. 1993), 1157–1170.

[14] Blum, B. I. *Software Engineering: A Holistic View.* Oxford University Press, New York, NY, 1992.

[15] Bridge, N., and Miller, C. Orthogonal defect classification: Using defect data to improve software development. *Software Quality 3* (Aug. 1997).

[16] Butcher, M., Munro, H., and Kratschmer, T. Improving software testing via odc: Three case studies. *IBM Systems Journal 41* (2002), 31–44.

[17] Card, D. N., and Glass, R. L. *Measuring Software Design Quality.* Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[18] Chaar, J., Halliday, M., Bhandari, I., and Chillarege, R. In-process evaluation for software inspection and test. *IEEE Trans. on Software Engineering 19*, 11 (Nov. 1993), 1055–1070.

[19] Chillarege, R. Orthogonal defect classification. In *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. McGraw-Hill, New York, 1995, pp. 359–400.

[20] Chillarege, R. Understanding Bohr-Mandel bugs through ODC Triggers and a case study with empirical estimations of their field proportion. *2011 IEEE Third International Workshop on Software Aging and Rejuvenation (WoSAR)* (January 2011), 7–13.

[21] Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., and Wong, M.-Y. Orthogonal defect classification — a concept for in-process measurements. *IEEE Trans. on Software Engineering 18*, 11 (Nov. 1992), 943–956.

[22] Crosby, P. B. *Quality Is Free: The Art of Making Quality Certain.* New York, McGraw Hill, 1979.

[23] Denning, P. J. What is software quality? *Communications of the ACM 35*, 1 (Jan. 1992), 13–15.

[24] Ellison, R., Nichols, W., and Woody, C. Measuring software assurance. *2016 IEEE 40th Annual Computer Software and Applications Conference* (2016), 359–364.

[25] Fenton, N. E., and Ohlsson, N. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. on Software Engineering 26*, 8 (Aug. 2000), 797–814.

[26] Garvin, D. What does 'product quality' really mean? *Sloan Management Review 26* (1984), 25–45.

[27] Geng, R., Chen, M., and Tian, J. In-process usability problem classification, analysis and improvement. In *Proc. 14th International Conference on Quality Software (QSIC 2014)* (Oct. 2014).

[28] Ghezzi, C., Jazayeri, M., and Mandrioli, D. *Fundamentals of Software Engineering*, 2nd ed. Prentice Hall, Englewood Cliffs, NJ, 2003.

[29] Gittens, M., Lutfiyya, H., and Bauer, M. An extended operational profile model. *ISSRE 2004 15th International Symposium on Software Reliability Engineering* (2004), 314–325.

[30] Gočeva-Popstojanova, K., and Trivedi, K. S. Failure correlation in software reliability models. *IEEE Trans. on Reliability 49*, 1 (Mar. 2000), 37–48.

[31] Goel, A., and Okumoto, K. Time-dependent error-detection rate model for software reliabiltiy and other performance measures. *IEEE Transactions on Reliability R–28* (August 1979), 206–211.

[32] Gokhale, S., and Trivedi, K. Log-logistic software reliability growth model. *Third IEEE International High-Assurance Systems Engineering Symposium Proceedings* (1998), 34–41.

[33] Hatton, L. Exploring the role of diagnosis in software failure. *IEEE Software 18*, 4 (July 2001), 34–39.

[34] Henderson, C. Managing software defects: Defect analysis and traceability. *ACM SIGSOFT Software Engineering Notes 33* (2008).

[35] Huang, L., Ng, V., Persing, I., Geng, R., Bai, X., and Tian, J. AutoODC: Automated generation of orthogonal defect classifications. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (Nov. 2011).

[36] Humphrey, W. S. *Managing the Software Process.* Addison-Wesley, Reading, MA, 1989.

[37] IEEE. *Reliability Handbook*. IEEE, New York, NY, 1984.

[38] IEEE. *IEEE Std 1633-2008, IEEE Recommended Practice on Software Reliability*. IEEE, 2008.

[39] IEEE. *828–2012 – IEEE Standard for Configuration Management in Systems and Software Engineering – Redline*. IEEE, 2012.

[40] ISO. *ISO/IEC 9126-1:2001 Software Engineering – Product Quality – Part 1: Quality Model*. ISO, 2001.

[41] Jeske, D. R., Zhang, X., and Pham, L. Adjusting software failure rates that are estimated from test data. *IEEE Trans. on Reliability 54*, 1 (Mar. 2005), 107–114.

[42] Kanoun, K., and Laprie, J.-C. Software reliability trend analyses from theoretical to practical considerations. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 20* (1994), 740–747.

[43] Kanoun, K., and Laprie, J.-C. Trend analysis. In *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. McGraw-Hill, New York, 1995, pp. 401–437.

[44] Keiller, P. A., and Mazzuchi, T. A. Enhancing the predictive performance of the Goel-Okumoto software reliability growth model. In *Proc. Annual Reliability and Maintainability Symposium* (Los Angeles, California, Jan. 2000), pp. 106–112.

[45] Keiller, P. A., and Mazzuchi, T. A. Investigating a specific class of software reliability growth models. In *Proc. Annual Reliability and Maintainability Symposium* (Seattle, Washington, Jan. 2002), pp. 242–248.

[46] Kitchenham, B., and Pfleeger, S. L. Software quality: The elusive target. *IEEE Software 13*, 1 (Jan. 1996), 12–21.

[47] Kumaresh, S., and Baskaran, R. Defect analysis and prevention for software process quality improvement. *International Journal of Computer Applications* (October 2010).

[48] Leszak, M., Perry, D., and Stoll, D. A case study in root cause defect analysis. *Proceedings of the 22nd International Conference on Software Engineering* (2000).

[49] Littlewood, B., and Verrall, J. L. A Bayesian reliability growth model for computer software. *Applied Statistics 22*, 3 (1973), 332–346.

[50] Lu, H., Kocaguneli, E., and Cukic, B. Defect prediction between software versions with active learning and dimensionality reduction. *2014 IEEE 25th International Symposium on Software Reliability Engineering* (2014), 312–322.

[51] Lutz, R. R., and Mikulski, I. C. Empirical analysis of safety-critical anomalies during operations. *IEEE Trans. on Software Engineering 30*, 3 (Mar. 2004), 172–180.

[52] Lutz, R. R., and Mikulski, I. C. Ongoing requirements discovery in high-integrity systems. *IEEE Software 21*, 2 (Mar. 2004), 19–25.

[53] Lutz, T. R., and Mikulski, C. Better analysis of defect data at NASA. *15th International Conference on Software Engineering and Knowledge (SEKE '03)* (2003), 607–611.

[54] Lyu, M. Software reliability engineering: A roadmap. *2007.FOSE '07 Future of Software Engineering* (2007), 153–170.

[55] Lyu, M. R., Ed. *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, 1995.

[56] Ma, L., and Tian, J. Web error classification and analysis for reliability improvement. *Journal of Systems and Software 80*, 6 (June 2007), 795–804.

[57] Malaiya, Y. K., Karunanithi, N., and Verma, P. Predictability of software-reliability models. *IEEE Trans. on Reliability 41* (1992), 539–546.

[58] Malaiya, Y. K., Li, M. N., Bieman, J. M., and Karcich, R. Software reliability growth with test coverage. *IEEE Trans. on Reliability 51*, 4 (Dec. 2002), 420–426.

[59] Malaiya, Y. K., Li, N., Bieman, J., Karcich, R., and Skibbe, R. The relationship between test coverage and reliability. In *Proceedings of fifth International Symposium on software reliability engineering* (1994).

[60] Midha, A. Software configuration management for the 21st century. *Bell Labs Technical Journal* (1997), 154–165.

[61] Midha, A. A case study in root cause defect analysis. *Proceedings of the 22nd International Conference on Software Engineering* (2000), 428–437.

[62] Moore, J. *Knowledge Area: Software Configuration Management*. Wiley-IEEE Press, 2006.

[63] Morozov, V., Kalnichenko, O., Timinsky, A., and Liubyma, I. Projects change management in based on the projects configuration management for developing complex projects. *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)* (September 2017), 939–941.

[64] Musa, J. D. Tools for measuring software reliability. *IEEE Spectrum* (1989), 39–42.

[65] Musa, J. D. Operational profiles in software reliability engineering. *IEEE Software 10*, 2 (Mar. 1993), 14–32.

[66] Musa, J. D., and Everett, W. W. Software-reliability engineering: Technology for the 1990s. *IEEE Software 7*, 6 (Nov. 1990), 36–43.

[67] Musa, J. D., Fuoco, G., Irving, N., Kropfl, D., and Juhlin, B. The operational profile. In *Handbook of Software Reliability Engineering*, M. R. Lyu, Ed. McGraw-Hill, New York, 1995, pp. 167–216.

[68] Musa, J. D., Iannino, A., and Okumoto, K. *Software Reliability: Measurement, Prediction, Application.* McGraw-Hill, New York, 1987.

[69] Musa, J. D., and Okumoto, K. A logarithmic Poisson execution time model for software reliability measurement. In *Proc. 7th Int. Conf. on Software Engineering* (Orlando, FL, Mar. 1984), pp. 230–238.

[70] Nelson, E. Estimating software reliability from test data. *Microeletronics and Reliability 17* (January 1978), 67–73.

[71] Oivo, M., and Basili, V. R. Representing software engineering models: The TAME goal oriented approach. *IEEE Trans. on Software Engineering 18*, 10 (Oct. 1992), 886–898.

[72] Pan, T., Leina, Z., and Chengbin, F. Defect tracing system based on orthogonal defect classification. *International Conference on Computer Science and Software Engineering* (2008).

[73] R, N., M.Fauzi, S. S., and Nasir, M. H. N. M. The development of software configuration management repository. *2009 International Conference on Information Management and Engineering* (April 2009), 423–426.

[74] Roehm, T., Bruegge, B., Hesse, T.-M., and Paech, B. Towards identification of software improvements and specification updates by comparing monitored and specified end-user behavior. *2013 IEEE International Conference on Software Maintenance* (2013), 464–467.

[75] Schneidewind, N. F. Software reliability model with optimal selection of failure data. *IEEE Trans. on Software Engineering 19*, 11 (Nov. 1993), 1095–1104.

[76] Shukla, R., Carrington, D., and Strooper, P. An extended operational profile model. *ISSRE 2004 15th International Symposium on Software Reliability Engineering* (2004), 314–325.

[77] Sylemez, M., and Tarhan, A. Using process enactment data analysis to support orthogonal defect classification for software process improvement. *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement* (October 2013), 120–125.

[78] Tian, J., Lu, P., and Palma, J. Test-execution-based reliability measurement and modeling for large commercial software. *IEEE Transactions on Software Engineering 21* (May 1995), 405–414.

[79] Ullah, N., Morisio, M., and Verto, A. A comparative analysis of software reliability growth models using defects data of closed and open source software. *IEEE 35th Software Engineering Workshop* (2012), 187–192.

[80] Vatanasombut, B., Stylianou, A. C., and Igbaria, M. How to retain online customers. *Communications of the ACM 47*, 6 (June 2004), 65–69.

[81] von Mayrhauser, A. *Software Engineering: Methods and Management*. Academic Press, San Diego, CA, 1990.

[82] Yamada, S., Ohba, M., and Osaki, S. S-shaped reliability growth modeling for software error detection. *IEEE Trans. on Reliability 32*, 5 (Dec. 1983), 475–478.