Reliability Estimation of Open Source Software based Computational Systems

Thesis submitted to

Cochin University of Science and Technology

in partial fulfillment of the requirement

for the award of the Degree of

Doctor of Philosophy in Engineering

Under the Faculty of Engineering

By

SHELBI JOSEPH

(Reg No. 3382)

Under the Guidance of

Dr. JAGATHY RAJ V.P

School of Engineering

Cochin University of Science and Technology

Cochin -682022

June 2014

Reliability Estimation of Open Source Software based Computational Systems

Ph.D. Thesis under the Faculty of Engineering

Author

Shelbi Joseph
Research Scholar
School of Engineering
Cochin University of Science and Technology
Kochi - 682022
Email: shelbi@cusat.ac.in

Supervising Guide

Prof. (Dr.) Jagathy Raj V. P.
School of Engineering
Cochin University of Science and Technology
Kochi - 682022
Email: jagathyrajvp@gmail.com

Co- Guide

Dr. P. S. Sreejith
Professor
School of Engineering
Cochin University of Science and Technology
Kochi - 682022

June 2014

School of Engineering
Cochin University of Science and Engineering
Cochin -682022

Certificate

Certified that the work presented in this thesis entitled **"Reliability Estimation of Open Source Software based Computational Systems"** is a bonafide work done by Mr. Shelbi Joseph under my supervision and guidance in the Division of School of Engineering , Cochin University of Science and Technology and that this work has not been included in any other thesis submitted previously for the award of any degree.

Dr. Jagathy Raj V. P
(Supervising Guide)

Dr. P.S. Sreejith
(Co- Guide)

Kochi- 22
Date:

Declaration

I hereby declare that the work presented in this thesis entitled **"Reliability Estimation of Open Source Software based Computational Systems"** is based on the original work done by me under the guidance of Dr. Jagathy Raj V.P, Professor, School of Management Studies, Cochin University of Science and Technology, and that this work has not been included in any other thesis submitted previously for the award of any degree.

Kochi- 22 **Shelbi Joseph**
Date:

In memory of my father……

Acknowledgements

At the outset, I thank ***God Almighty*** *for providing me the great opportunity to do this research work and complete it within the stipulated time. It is only He who has led me to the most appropriate personalities for the same, and made me withstand the stress during the course, without my losing confidence and motivation.*

I have great pleasure in expressing my profound gratitude to my guide, ***Dr.Jagathy Raj V.P.****, Professor, Cochin University of Science and Technology, for the motivation and guidance provided throughout this research work. I appreciate his approachability and the trust bestowed on my research efforts. Consistent support and sincere encouragement provided throughout the period are gratefully acknowledged.*

I wish to express my sincere gratitude to ***Dr. Sreejith P.S.****, Professor and Principal, School of Engineering, Cochin University of Science and Technology, for his guidance, help and encouragement throughout the period of this work. He has been my co-guide and friend as well.*

I am highly indebted to ***Dr. G Santhosh Kumar****, Assistant Professor, Cochin University of Science and Technology and* ***Dr. P. V. Shouri*** *, Associate Professor, Model Engineering College , Cochin for helping me to conceive the appropriate approach to the whole work and contributing significantly to the formulation of the precise methodology. Only because of their willingness to share their knowledge and having fruitful and meaningful discussions with me, I could complete this research work in such a good manner. Sparing innumerable collection of literature and the whole hearted involvement, demonstrated an incredible level of perseverence and affirmation. I gratefully acknowledge their*

willingness to spare valuable time to make this thesis, a technically correct one. Their systematic and perfect way motivated and helped me to improve my academic and personal qualities.

I take this opportunity to acknowledge the support and assistance extended by ***Dr. David Peter****, Professor, School of Engineering, and for the promptness and sincerity offered in dealing with things.*

I am thankful to ***Dr. Gopikakumari****, Professor and Head, Division of Electronics and Communication Engineering, School of Engineering, and also my doctoral committee member, for giving valuable guidance at the time of interim presentations.*

I have no words to express my deep sense of gratitude to my colleagues and well-wishers at Information Technology Division, School of Engineering, Cochin University of Science and Technology, for helping me to complete this effort, by offering generously to share my work at the college.

I express deep gratitude towards ***Sariga Raj****, research scholar, Cochin University of Science and Technology and* ***Renumol .V.G****, research scholar, IIT Madras, for the efforts they have spared in making the numerous collection of literature available to me from time to time, during the entire period.*

I gratefully remember the support and encouragement extended by ***Dr. K.A.Zakkariya****, Associate Professor, School of Management Studies. His timely suggestions made me do things effectively. I thank him at this moment.*

My sincere thanks to all non teaching staff of my department for their cordial relations, sincere co-operation and valuable help, especially so, to Shiji.S.H and Shibu A.S.

I thank all the faculty members of the School of Engineering, for the co-operation extended towards me.

I also wish to place on record my sincere thanks to my parents and family for the co-operation and support extended towards me.

Finally I wish to place on record my sincere thanks to one and all, who have helped me in the completion of this work.

Shelbi Joseph

Abstract

Software systems are progressively being deployed in many facets of human life. The implication of the failure of such systems, has an assorted impact on its customers. The fundamental aspect that supports a software system, is focus on quality. Reliability describes the ability of the system to function under specified environment for a specified period of time and is used to objectively measure the quality. Evaluation of reliability of a computing system involves computation of hardware and software reliability. Most of the earlier works were given focus on software reliability with no consideration for hardware parts or vice versa. However, a complete estimation of reliability of a computing system requires these two elements to be considered together, and thus demands a combined approach. The present work focuses on this and presents a model for evaluating the reliability of a computing system. The method involves identifying the failure data for hardware components, software components and building a model based on it, to predict the reliability. To develop such a model, focus is given to the systems based on Open Source Software, since there is an increasing trend towards its use and only a few studies were reported on the modeling and measurement of the reliability of such products. The present work includes a thorough study on the role of Free and Open Source Software, evaluation of reliability growth models, and is trying to present an integrated model for the prediction of reliability of a computational system. The developed model has been compared with existing models and its usefulness of is being discussed.

Key Words: *Failure rate, hardware reliability, mean time between failures, open source software, software reliability*

Preface

The reliability of computation systems involves, failure-free operation of the software as well as hardware components. A lot of software models are available, and Software Reliability Engineering (SRE) is a skill to be more competitive in the environment of globalization and outsourcing. Customers want a more reliable software that is faster and cheaper. SRE is a practice that is standard, proven, and widely applicable. It is low in cost and its implementation has virtually no schedule impact (Musa [2005]). The hardware reliability can be evaluated from the available field failure data of the components. Constant Hazard model is widely used in the literature for evaluation of hardware reliability.

Software development is very competitive and there are a lot of developers in a given domain. It is not sufficient that a software works, but it is important that it meets the customer-defined criteria. Surveys reveal that the most important quality characteristics are reliability, availability, rapid delivery and low cost. These are primarily user-oriented rather than developer-oriented attributes. A large sampling of software developers indicates that the most important problem facing them, is how to resolve conflicting demands that customers place on them for important quality characteristics of software based systems. For a long time quantitative measures have existed for delivery time and cost. Reliability suffers when it competes for attention against schedule and cost. In fact, this may be the principal reason for well known existence of reliability problems in many software products. Engineering software reliability involves developing a product in such a way that the product reaches the market at the right time, at an acceptable cost and with expected reliability.

The SRE process consists of defining the product, implementing operational profiles, and engineering the just right reliability. The cost of applying SRE is low and its impact on schedule is minor. However, the benefits are large. The aim of software reliability engineering is to model the failure behavior of software systems to estimate and forecast reliability. Software reliability estimating serve many purposes. For instance, software reliability estimates can allow a contractor and a buyer to contractually agree to some tangible measure of reliability performance that a software system is expected to achieve. Also, software reliability estimates can allow software users to be selective about the software they purchase by considering the advertised software reliability (Jean and Terry [1995]).

The development of integrated hardware-software reliability is very difficult. Mark and Christine [1995] brought out some of the differences between hardware and software reliability modeling which make integrating together very difficult. The present work assumes significance in that, a simplified model has been proposed that incorporates both hardware and software reliabilities.

The proposed thesis presented in eight chapters, deals with the work carried out in designing and developing Reliability Estimation of Open Source Based Computational Systems by Integrating Hardware and Software Components.

The thesis is organized as follows:

Chapter 1 introduces the area of reliability and open source software.

Chapter 2 is a systematic survey of existing reliability models used in the industry for hardware and software (both closed source as well as open

source software). It also mentions some new techniques and technologies for measuring and improving software reliability and a frame work to enable the early prediction of software reliability, incorporating reliability measurement in each stages of the software development.

Chapter 3 discusses the scheme of research work and methodology which involves studying the effects of failure of actual software packages and working towards formulating a reliability model taking into consideration the hardware issues.

Chapter 4 elaborates the study and analyzes the role of Free and Open Source Software (FOSS) in different communities.

Chapter 5 focuses on the study and evaluation of existing open source software and arrives at a reliability growth model.

Chapter 6 formulates an algorithm for estimating software reliability and the development of a simplified model.

Chapter 7 details the evaluation and comparison of the developed model with the application of a simplified model and its application in real time situation.

Chapter 8 recapitulates the thesis and mentions conclusions and research findings. Some of the results have been published in international journals and in the proceedings of various national and international conferences.

Contents

Chapter 3

Chapter 4

List of Tables

List of Figures

Acronyms

Software Reliability Engineering (SRE)
Open Source Software (OSS)
Free and Open Source Software (FOSS)
Information Technology (IT)
Hardware Reliability Growth Model(HRGM)
Information Technology Enabled Services (ITES)
Computational Annual Growth Rate(CAGR)
Closed Source Software (CSS)
Open Source Observatory and Repository(OSOR)
Software Reliability Growth Models (SRGMs)
Non Homogeneous Poisson Process(NHPP)
Goel-Okumoto NHPP Model (G-O)
Software Reliability Growth Model (SRGM)
Musa-Okumoto Logarithmic Poisson Model(M-O)
The Indian software and Information Technology Enabled Services (ITES)
Compounded Annual Growth Rate (CAGR)
Cumulative Distribution Function (CDF)
Mean Time To Failure (MTTF)
Bug Tracking System (BTS)
Mean Time Between Failures (MTBF)
Reliability Block Diagram (RBD).
Failure Density (f_d)
Failure Rate (Z)
Reliability (R)

Chapter 1

Introduction

Contents

1.1 Introduction

The quality of a software product decides its acceptance or fate in the software development life cycle. High developmental costs and increasing global competition have intensified the pressures to quantify software systems quality, and the need to measure and control the level of quality delivered. Reliability is the most important and most measurable aspect of software systems quality, and is customer- oriented. It is the capability of a system to deliver results accurately every time the user

requests it. The performance of a system is largely affected by its failure to deliver or downtime. Therefore, a system is considered to be reliable if it can rectify its failure at minimum time, thereby ensuring guaranteed results to the user. It is a measure of how well the product functions, to meet its operational requirements. In other words, it decides the software product's acceptance or fate in the life cycle. It ensures the products capability to rectify its failure.

In a computer system, hardware and software are interdependent. Without hardware, the software system is an abstraction, which is simply a representation of some human knowledge and ideas. Without software, hardware is a set of inert electronic devices. However when they are put together to form a system, a machine is created that can carry out complex computation, and deliver the results of these computations to its user. Systems have properties that only become apparent when their components are integrated and operated together. Hardware failure can generate spurious signals that are outside the range of inputs expected by the software. The software can then behave unpredictably and produce unexpected outputs. So, a hardware failure may lead to software problems that could overload the hardware, causing more failures. Thus the initial failure which might be recoverable, can rapidly lead to developing into a serious problem that may result in the complete shutdown of the system.

Until the late 1960's, attention was almost solely on hardware related performance of the system. In the early 1970's, software also

became a matter of concern, primarily due to a continuing increase in the cost of software relative to hardware, in both the development of the system and its operational phases.

Many models where put forward to address the reliability of the computer system, considering software and hardware components independently. The total performance of the system can be modeled only by considering these components together. Many successful software products were developed following Free and Open Source Software Development (FOSS) methodology. The development model used in this scenario are entirely different from traditional software development (closed source). So, the reliability models developed for closed source software cannot be used for FOSS, moreover, the dearth of valuable models call for studies in this area.

This study tries to put forward a model for the estimation of the reliability of a computer system, by integrating hardware and software components, especially FOSS.

In this chapter, the basic notions of reliability is introduced, reliability approaches during software life cycle phases, along with discussions on consequences of reliability and its impact on software industry. The research motivation and objectives are listed further, followed by the outline of the thesis.

1.2 Reliability

Reliability is a measure of continuous delivery of the correct service or equivalent of the time of failure (Jean and Terry [1995]).

Software being a complex intellectual product, some errors are inevitable during requirements formation as well as during designing, coding, and testing the product. The development process for high-quality software includes measures that are intended to discover and correct the faults resulting from these errors, including reviews, audits, screening by language-dependent tools, and several levels of tests. Managing these errors involves describing the errors, classifying the severity and criticality of their effects and modeling the effects of the remaining faults in the delivered product, and thereby working with designers to reduce their number of errors and their criticality(IEEE Std 1413-[2010]).

The reliability definitions given in the literature vary between different practitioners as well as researchers. The generally accepted definition is as follows:

Definition: Reliability is the probability of success or the probability that the system will perform its intended function under specified design limits. More specifically, reliability is the probability that a product or part will operate properly for a specified period of time (design life) under the design operating conditions such as temperature, voltage etc., without failure. In other words, reliability may be used as a measure of the system's success in providing its function properly. Reliability is one of the quality characteristics that consumers require from the manufacturer of products.

Reliability can also be defined as the probability that an item can perform a required function for a specified period of time under the specified operating conditions (Kumar et. al. [1992], Goel et. al. [2002], Patrick [2002] , Charles [2000], Srinath [1991]). This definition has four key elements: The quantification of reliability in terms of *probability*. A statement defining the *required function* – as the function is defined in detail, it becomes more clear, which product failures impair the success of the mission and which do not. A statement specifying the *period of time* – deterioration of materials and parts with time is natural, and consequently the performance level of the unit will also go down with time. If the time period is not specified, probability is a meaningless number for time oriented products, and a simpler statement defining the operating condition

Product failures cost money has an impact on the development schedules and system performance through the increased use of computer resources such as memory, CPU time, and peripheral requirements. Consequently, there can be too much as well as too little effort spent dealing with faults. The system engineer along with management can use reliability estimation and assessment to understand the current status of the system and make trade-off decisions. The basic objective is to identify required elements for an understandable, credible reliability prediction, which will provide sufficient information to the users to evaluate the effective use of the prediction results. A reliability prediction should have sufficient information concerning inputs, assumptions, and

uncertainty, so that the risk associated with using the prediction results would be understood. To analyze the reliability characteristics further, it is necessary to look at the hardware and software reliabilities separately.

1.3 Hardware Reliability

Hardware reliability is nothing but the ability of hardware to perform its functions for some specific duration of time and is expressed as mean time between failures (MTBF). Computer systems, whether hardware or software, are subject to failure. A failure may be produced in a system or product when a fault is encountered resulting in the non operation or disability of the required function and a loss of the expected service to the user (Norman [2008]).The field of hardware reliability has been established for some time, which is related to software reliability and the division between hardware reliability and software reliability is somewhat artificial and both may be defined in the same way. Therefore, it is possible to combine both hardware and software component reliabilities to get system reliability (Musa [1980]).

Usually, hardware design failures are low because hardware is generally less complex than software. Engineers have not applied the reliability concepts to hardware to any extent. The probability of failure due to wear and tear and other physical causes has usually been much greater, than failures due to an unrecognized design problem. Hardware design failures had to be kept low because, retrofitting of manufactured items in the field was very expensive. The advancement of research work has shown

that parallels can be drawn between software engineering and chip design. This is mainly attributed to the realization of the importance of relationship between software reliability and hardware reliability (Musa [1980], Shooman [1986] and Lloyd and Lipow [1977]).

1.3.1 Reasons for Hardware Failure

The risk of hardware failure is the most commonly talked-about reason to perform backups. The worst kind of failure is the unrecoverable hard disk failure as the hard disk is the main storage of a system. However, there are other hardware problems that can cause permanent data loss, and some of these can be rather hard to figure out. Memory errors, system timing problems, resource conflicts and power loss are some of them (www.pcguide.com/care/bu/risksHardware-c.html).

Both software and hardware reliabilities depend on the environment. The source of failure in hardware is physical deterioration, whereas, that in software is design faults. The concepts and theories developed for software reliability could be really applied to any design activity including hardware design. Once a software design defect is properly fixed, it is in general fixed for all time. Failure usually occurs only when a program design is exposed to an environment that was not tested or developed. Although manufacturing can affect the quality of physical components, the replication process for software design is trivial, and can be performed to very high standards of quality (Musa [2005]).

1.4 Software Reliability

The IEEE defines software reliability as the probability that software will not cause the failure of a system for a specified time under specified conditions [IEEE Std 982.2-1988]. Software reliability denotes the probability that a software product in a pre-defined condition performs its tasks without malfunctioning for a specified period of time.

Software Reliability is important for many sectors of the software industry. Besides knowing how to achieve reliability, the most important thing is to know the actual reliability achieved in a specific software product. Assessing the reliability of software- based systems is increasingly necessary because of the survival of companies and at times the lives and limbs of people on the service they expect from the software. Sound decision-making requires some understanding of the uncertainties thus incurred. Meanwhile, software complexity increases and progress in development tools enables more lesser- trained people to build software-based systems. The short term economic incentive to use off-the-shelf software, even in sensitive applications, imposes new requirements to evaluate the risk thus assumed. The pressure on vendors to guarantee some level of quality of service will thus also increase, extending from bespoke software to off-the-shelf software and from mission-critical to productivity-critical software.

A Software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it; fault introduction, fault removal, and the operational environment. The

failure rate of a software system generally decreases due to fault identification and removal. It is possible to observe the history of failure rate by statistically estimating the parameters associated with the selected model. The purpose of the model is twofold. Firstly, to estimate the extra execution time during the test required to meet a specified reliability objective and secondly to identify the expected reliability of the software when the product is released (IEEE Std 1413-[2010]).

Apart from classical hardware reliability, software reliability has a different nature (Rosenberg and Hammer [1998], Musa [1975]). While the reliability of hardware continues to change even after the product is delivered, the reliability of software is improved throughout the development process, until the product is delivered.

After the delivery, a change in reliability level is possible only if maintenance action is performed to either compensate for defects in the software or to catch up with technological advances. Another major difference between software reliability and hardware reliability is that software reliability is not a function of how frequent that specific software is used, whereas hardware is subject to wear out (Musa [1975]). Moreover, software being conceptual, documentation is considered as an integral part of software and software reliability.

1.4.1 Reasons for Software Failure

Octavio [2008] suggests the reasons based on his professional experience, to explain what causes software failure. According to him,

the main reasons are software complexity, commercial deadlines and market competence, competences and skills of the teamwork, underestimating good software engineering design, and poor quality control. Software complexity is increased when a software application is conceptually flooded with a lot of features. The Windows Vista was delayed because of its inherent complexity and over ambition to incorporate a lot of features.

Commercial deadlines and market competence are odd factors that impact software development. Most of the software commercially released, goes to the market prematurely in order to catch customer mindshare, achieve a competitive advantage and earn market share. Unfortunately, premature go-to-market may hurt the possibility of any software application by being competitive in the long term in an aggressive and crowded market.

Competencies and skills of the teamwork are two of the most important factors that are needed to succeed in management activities and development issues typically found in big software development projects, that are ambitious and highly complex.

One common pitfall with odd consequences in software development activities happens with an ill-devised design and a poor conceptual model to make savings and go ahead according to a wrongly designed schedule. By underestimating software engineering and design, it is very hard to develop a software project on schedule with world-class quality

and achieve satisfactory financial outcomes from a long term perspective. Usually, systematic and well developed beta testing of the software is cut down and minimized to privilege a faster release to market. Thus the overall quality of the software is seriously compromised, the customer base gets angry for the unresponsiveness of the product, and in the long term, financial profits may be hurt.

Error-free software release is practically impossible and really counter-economical due to the inherent complexity of the involved code and inherent design. Thus, the users are working as beta testers by using a software application prematurely released to the production environment.

Octavio [2008] states that software failures are due to unclear business requirements and Scope Creep. A project without proper business requirements and specifications are doomed to fail. It is either due to little information provided to the system analysts for them to come up with concrete business requirements and specifications, or lots of assumptions about the clients by the system analysts.

Scope creep is the state in which the customers always ask for some new requirement, some new feature and focus on it so much that it shifts the focus from the core requirements and off course the customer is the king and that the project starts lagging and they start realizing that core functions are still outstanding.

Dzumbu [2008], an Analyst at AEL Mining Services shares his experience with respect to software failure as less testing and assumptions.

Due to the demand from managers and owners to get the software deployed, and running the developers, usually deploy things without proper and extensive testing. At most, they do optimistic testing that does not address unfriendly hostile environments that are much more realistic than their test environments.

It should not be assumed that the user will follow certain steps in using the software. It is learned that the moment the user figures out a different route which is not anticipated, problems may be created. This of course boils down to testing, the test scenarios sometimes do not cover all possible scenarios and this leads to surprise behavior that can lead to the collapse of the whole system.

1.5 Open Source Software

Open Source Software (OSS) has created an interest in the software development circle. It is an emerging software development environment, where the design and development strategy is radically different from the closed development counter parts. The success of an OSS is often related to the number of developers it can attract. This larger community of developers called the 'Bazzar', identifies and eliminates software defects and adds more features through a peer-review process. The phase where the number of active developers and the actual work performed on the system is constant and is termed as the "Cathedral" (Capiluppi and Michlmayr [2007]).

Open source development is an area where people develop and distribute their products by downloading free source code available under a license. In closed source environment, the development process is a systematic approach following system study, design, coding, Testing and maintenance. In this the reliability estimation is based on stable programs that are not undergoing design changes and are completely integrated. (Musa and Iannino [1981]). But in reality it is not the case, as in custom developed software, it is difficult to have programs not undergoing design changes. Usually it is only after the first run, customers come to know of their actual necessities, so it is essential that design changes are needed. Further, it is assumed that all codes are being executed at one time or more, to make sure that the resulting requirements are met, but the customer may not be satisfied with these outcomes, and might need more. Another thing is that in the development phase, the outcome is correct in one way, but logically, it may not be the case.

Free and Open Source Software (FOSS) refers to those categories of software products that allow users to use, modify and redistribute the software without the need to pay a royalty fee to the author of the product[www.gnu.org, www.opensource.org]. FOSS product includes both system and application software like GNU/Linux, Apache Web Server, Postgresql, OpenOffice, Gimp, OrangeHRM etc. (Smrithy et. al. [2009]). Universities and colleges spend a huge sum on laboratories and software. The huge software installation costs can be cut down by using FOSS alternatives, instead of proprietary software [www.osalt.com].

Since a lot of people are working on OSS, and they are making use of it to develop their own required products and which is useful for the human beings, the importance of its quality and hence the reliability is an important factor to be considered. Literature study reveals that a lot of people are using OSS products for their daily life activities, but relevant works to model the reliability of OSS is not sufficient and hence an attempt to do so will be a useful work, and this work attempts to develop a model for the same.

1.6 Reliability Approaches within the Phases of Software Life Cycle Phases.

A competitive and mature software development organization targets a high reliability objective from the very beginning of software development. Generally, the software life cycle is divided into the following phases:

Requirements and definition: In this phase, the developing organization interacts with the customer organization to specify the software system to be built. Ideally, the requirements should define the system completely and unambiguously. In actual practice, there is often a need to do corrective revisions during software development. A review or inspection during this phase is generally done by the design team to identify conflicting or missing requirements. A significant number of errors can be detected by this process. A change in the requirements in the later phases can cause increased defect density (Cleland and King [1992]).

In the *Design phase*, the system is specified as an interconnection of units, such that each unit is well defined and can be developed and tested independently. The design is reviewed to recognize errors.

With the *Coding phase*, the actual program for each unit is written, generally in a higher-level language. Occasionally, assembly level implementation may be required for high performance or for implementing input-output operations. The code is analyzed in a team meeting to identify errors.

The *Testing phase* is a critical part of the quest for high reliability and can take 30%–60% of the entire development time. It is often divided into the following four sub phases. *Unit test*: In this phase of testing, each unit is separately tested, and changes are done to remove the defects found. As each unit is relatively small, and can be tested independently, they can be exercised much more thoroughly than a large program. *Integration testing*: During integration, the units are gradually assembled, and partially assembled subsystems are tested. Testing subsystems allows the interface among modules to be tested. By incrementally adding units to a subsystem, the unit responsible for a failure can be identified more easily. *System testing*: The system as a whole is exercised during system testing. Debugging is continued until some exit criterion is satisfied. The objective of this phase is, to find defects as fast as possible. In general, the input mix may not represent what would be encountered during the actual operation. *Acceptance testing*: The purpose of this test phase is to assess the system reliability

and performance in the operational environment. This requires collecting or estimating information on how the actual users would use the system. This is also called a-testing. This is often followed by b-testing, which involves the use of the b-version by the actual users.

Operational use and maintenance: Once the software developer has determined that an appropriate reliability criterion is satisfied, the software is released. Any bugs reported by the users are recorded, but are not fixed until the next patch or bug-fix. In case a defect discovered represents a security vulnerability, a patch for it needs to be released as soon as possible.

A general classification of software reliability models based on the software life cycle phases are as shown in Fig 1.2 (Sharma et. al. [2010], Popstajanova and Trivedi [2001], Pham [2003], Huang et. al. [2003], Smidts at. al[1998]). This is a generalized classification starting from the requirement analysis, design, implementation, testing and validation operations that exists, in the different phases of the software development process.

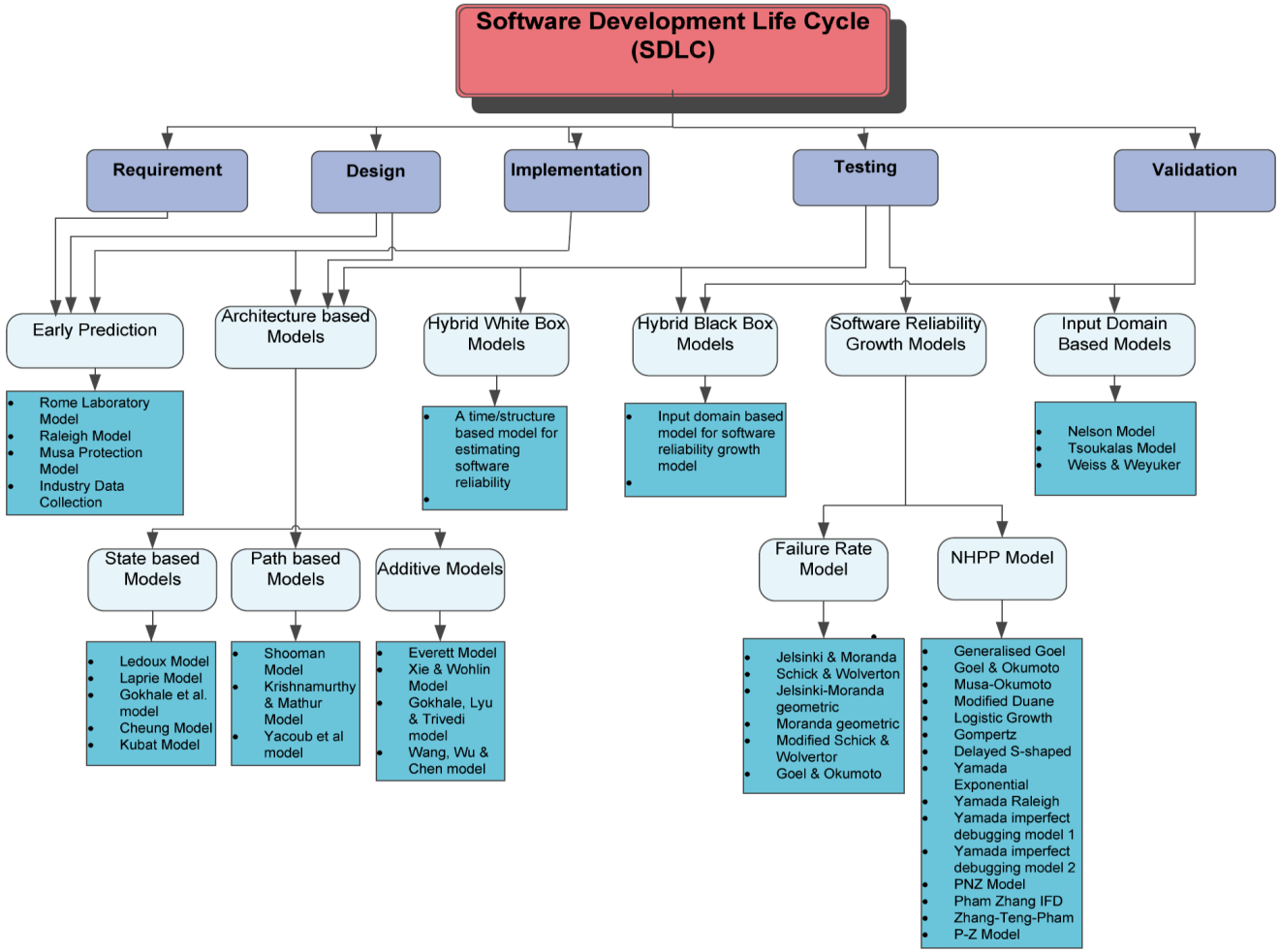


Figure 1.1 Classification of software reliability models based on software life cycle phases (Sharma et. al. [2010], Pham [2003], Huang et. al. [2003], Smidts at. al [1998]).

Reliability growth for software is the positive improvement of software reliability over time, accomplished through the systematic removal of software faults. The rate at which the reliability grows depends on how fast faults can be uncovered and removed. A software reliability growth model allows project management to track the progress of the software's reliability through statistical inference and to make projections of future milestones. If the assessed growth falls short of the planned growth, the management will have sufficient notice to develop new strategies, such as the re-assignment of resources to attack identified problem areas, adjustment of the project time frame, and re-examination of the feasibility or validity of requirements.

Measuring and projecting software reliability growth requires the use of an appropriate software reliability model, that describes the variation of software reliability with time. The parameters of the model can be obtained either from the prediction performed during the period preceding system test, or, from the estimation performed during the system test. Parameter estimation is based on the times at which failures occur.

The use of a software reliability growth-testing procedure to improve the reliability of a software system to the defined reliability goal implies that, a systematic methodology will be followed for a significant duration. In order to perform software reliability estimation, a large sample of data must be generated with a reasonable degree of confidence.

1.7 Consequence of Reliability and its Impact on Software Industry

The invention of computer system and its use in the daily life of human beings has created the advancement of the hardware as well as software in computer systems. Today, computer hardware and software permeates our modern society. The newest cameras, VCRs, and automobiles can not be controlled and operated without computers. Computers are embedded in wristwatches, telephones, home appliances, buildings, and aircraft. Today, technology demands high-performance hardware and high-quality software for making improvements and breakthroughs. Industries like automotive, avionics, oil, telecommunications, banking, semiconductor and pharmaceutics all rely on computers for their functioning.

The size and complexity of computer-intensive systems have grown dramatically and these can be found in projects under taken by NASA, aviation industry and telecom industries among others. NASA projects including space shuttle launching consumes approximately 500,000 lines of software code that is on board, and 3.5 million lines of code for ground control. Projects like these demand high accuracy and reliability with zero tolerance to faults [http://www.cse.cuhk.edu.hk/~lyu/book/reliability/introduction.html].

Similarly, the avionics industry extensively uses embedded software. It is a huge mix of hardware and software in action for processing each flight, landing, and take off. This again calls for accurate working of these

systems with no failures. Other fields of intensive computer systems are the telecom industry, catering to millions of users day and night. Fault free delivery of services is a major concern for all telecoms.

The concern arises mainly due to the unbalanced development of hardware and software. Even though it is the software that has the integrating potential which gives the designer the edge to design complex systems, it is this very software that might be responsible for the majority of failures. Though there has been rapid advancements in hardware technology, the proper development of software technology has failed to keep pace with it in all measures including quality productivity, cost and performance.

Software engineers, when determining the quality of the software, feel that software reliability is one of the important factors which affects the system performance. Software problems are the main causes of system failures today. There are many well-known cases of the tragic consequences of software failures. In critical systems, very high reliability is naturally expected. Studies have shown that reliability is regarded as the most important attribute by potential customers. All software developed will have a significant number of defects. All programs constituting the software must be tested and debugged, until a sufficiently high reliability is achieved. It is not possible to ensure that all the defects in a software system have been found and removed; however, the number of remaining bugs must be very small. For software systems, quantitative methods for achieving and measuring

reliability are coming in use, because of the emergence of well-understood and validated approaches. Enough industrial and experimental data are available to develop and validate methods for achieving high reliability. The defect or fault or bug refers to an error in system implementation that can cause a failure during execution. Defects with very low testability can be very difficult to detect. The software reliability improves during testing, as bugs are found and removed. Once the software is released, its reliability is fixed as long as the operating environment remains the same (Musa [1987]).

Over the past decade, information technology(IT) industry has become one of the fastest growing industries in India. Strong demand over the past few years has placed India amongst the fastest growing IT market in the Asia-Pacific region. The Indian software and information technology enabled services (ITES) industry has been a remarkable success story. It has grown at a compounded annual growth rate (CAGR) of 28 percent during the last few years (Subash [2006]).

The Indian software industry is more service oriented rather than product oriented. It is heavily export-oriented and is largely managed by professionals. Although the industry has grown in a spectacular fashion, sustaining this performance will pose a number of challenges, of which improving the reliability of the product is the most important one.

1.8 Need for Early Prediction of Software Reliability

Software intensive systems are influencing the development of all the facets of human society. The developmental activities of such systems are mainly performed in a labour-intensive way. Introduction of various faults in the software system are inevitable and may cause failure in near future. The impact of such failures may have critical consequences for infrastructure, economy or even the safety of human lives. Both the cost of software development and losses from its failure are expensive. Therefore, there is a growing need to ensure reliability of these software systems as early as possible. Prediction of faults in each stage of software development becomes important since earlier a fault is identified, the better and more cost-effectively it can be fixed. This means that the reliability of software systems is of primary importance.

Error prevention, fault detection and its removal are the major activities that follow to achieve reliability in software. There are many matrics proposed in literature to maximize the reliability of a software system specifically measures above mentioned activities (Vinay Tiwari and Pandey [2012]). Brooks [1995], has made an observation related to software development *as the total cost of maintaining a software system is typically 40 %or more of the cost of its development*. This observation demands the early detection of bugs and a model to predict the reliability of a software system. FOSS development is usually carried out by a community, the development pace and the quality depends on percentage of developers who are actually using the software. There is a chance

that most of the possible bugs are reported at the early stages of the development and most of them get fixed at the same time. This dynamics is important to predict quality of the system being developed. Data on this behaviour is readily available in various repositories and project development web sites. This will enable a quantitative approach towards measuring the reliability of FOSS.

Many organizations including Government are adopting FOSS based solutions to minimize the cost of the software. It is difficult to select an appropriate open source based software to the existing infrastructure without conducting a detailed study on the behaviour of the software. The reliability study proposed in this work will suggest some pointers in the decision making on selection of the appropriate software.

1.9 Integrated Software Hardware Reliability

Computer systems, whether hardware or software, are subject to failure. The classical reliability theory can be extended in order to be interpreted from both hardware and software viewpoints and is referred to as X-ware (Laprie and Kanoun [1992]). There are at least two significant differences between hardware reliability and software reliability. Firstly, a software does not wear out or fatigue. Secondly, due to the accessibility of software instructions within computer memories, any line of code can contain a fault that, upon execution, is capable of producing a failure. The development of an integrated hardware software reliability model is a difficult task. The model can be developed by

collecting the failure behavior of both software and hardware (Boyd and Monahan [1995]). The progressively increasing use of computer controlled systems, where software and hardware play an equally important role, is increasing. That is, the reliability of both software and hardware is important as the overall performance of the system. This is particularly significant to safety critical systems, such as some that may be found on commercial aircraft. Usually as per the literature, studies on hardware reliability or software reliability are carried out separately. That is, a separate analysis is done for each often. The main drawback here is that, there are no widely accepted standard methods for combining results of separate hardware and software reliability analyses together into a meaningful composite result. This is because, when a program is executed, its performance is dependent on both the underlying hardware as well as the software engineering methods adopted. Hence the best approach would be to use a method which models both hardware and software reliability in one integrated system model.

The development of system reliability models which accurately represent the failure behavior of both hardware and software components, is a difficult task. The job is complicated by the fact that the failure processes of hardware and software are intrinsically different. Furthermore, the topic of how to accurately model software reliability and hardware reliability and to integrate it into a system reliability aspect, is difficult. The main reason is that the failure

processes for hardware and software are completely different in nature. Increased use of firmware and embedded software is blurring the boundary line between software and hardware. Software is not hardware, software does not break or wear out over time like hardware (Debra and David[1999]).

1.10 Motivation

In the past few decades there is a dramatic growth in size and complexity of software systems developed. The reliability of the systems is becoming more important, as the failures of such systems impact heavily on the business performance. Despite the body of knowledge evolved in the area of reliability, challenges and open questions do still exist. There are several models developed to explain the reliability of the end product, but not considering each stages of the process models followed. Developing reliability models, incorporating failure data from each stage of the process, will accurately predict the reliability of the software developed.

There is an obvious trend towards the adoption of FOSS even for critical applications such as data centers and commercial aircraft. It will be interesting to study why various sectors of information technology are lining up behind the use of FOSS and the benefits of adoption of FOSS for building their custom products. The reliability study of FOSS systems are attracted researchers recently and only a few models were suggested. The failure data of many FOSS systems are available openly

and can be utilized to develop a model to increase the reliability of the software that is being developed.

Reliability theories developed over the years have successfully allowed hardware systems to be built with high reliability requirements and the final system reliability to be evaluated with acceptable accuracy. In recent years, however, many of these systems have come to depend on software for their correct functioning. So the reliability of software has become more and more important. As the software is becoming increasingly complex, different models are required to study the reliability of such software systems. Further, it is important that the software reliability be integrated with hardware as a computing system, which is made up of both these components.

1.11 Objectives

The existing reliability models have been studied and an attempt has been made to develop a new methodology towards measuring and improving software reliability. Existing reliability models do not address reliability aspects in all stages of software development. It is difficult to predict the reliability of the end product. The proposed work is an attempt to make such a frame work to enable early prediction of software reliability incorporating reliability measurement in each stage of the software development. The main objective of the research is to develop a model to represent reliability of a computing system by considering both hardware and software failure impacts. More specifically, the objectives of the research are as follows:

1) To study the existing reliability models
2) To explore techniques and technologies for measuring and improving software reliability.
3) To develop a framework to enable the early prediction of software reliability incorporating reliability measurement in each stage of the software development.
4) To study and analyze the role of Free and Open Source Software (FOSS) in different communities.
5) To study and evaluate existing open source software and to arrive at a reliability growth model.
6) To develop a model for estimation of computational reliability by incorporating both software and hardware components.
7) To evaluate and compare the actual reliability with the developed model and other existing models.

1.12 Methodology

The research work that is to be carried out for accomplishing the objectives as mentioned earlier, follows the methodology as depicted in Fig 1.2.

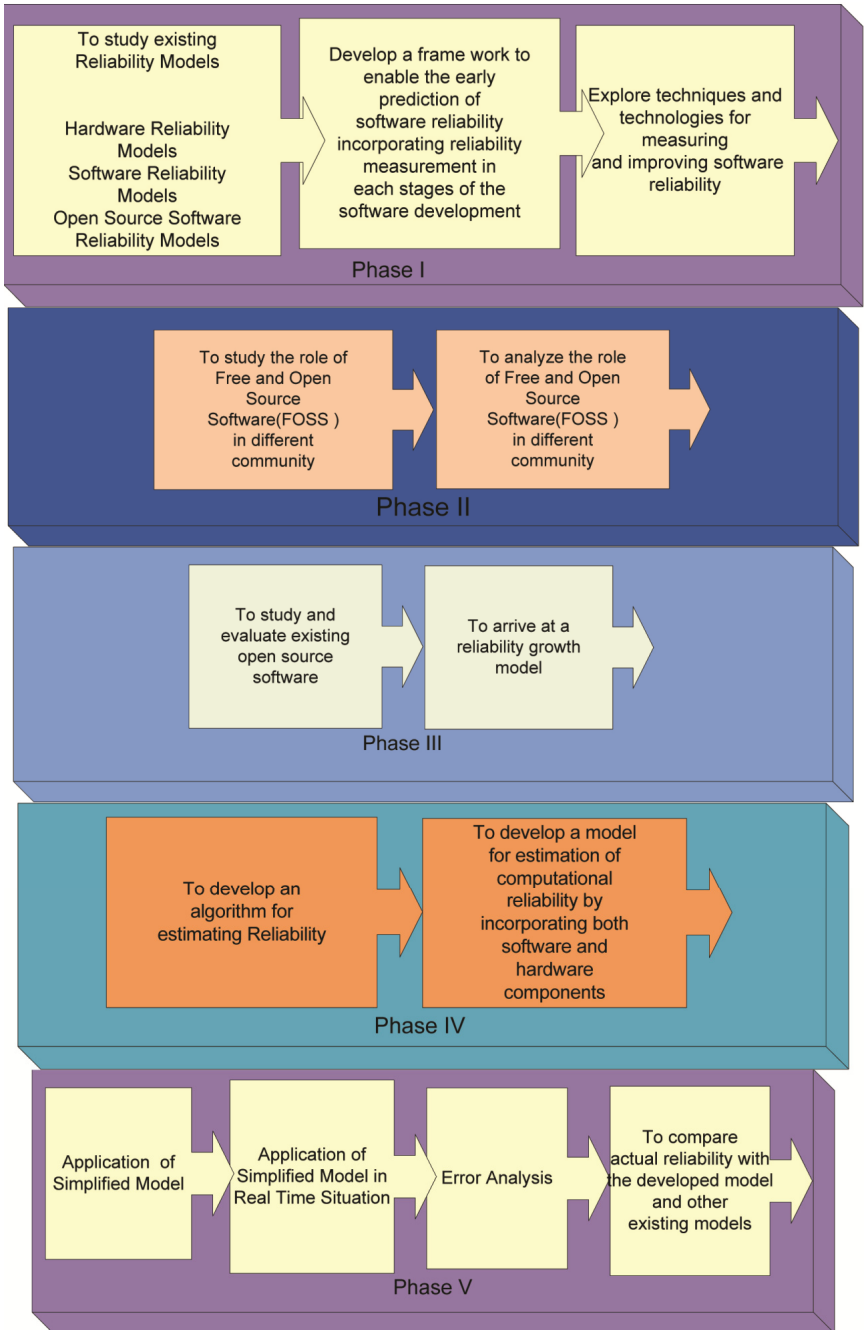


Figure 1.2 Research Methodology

The first phase of the work is the study of the existing reliability models. In this phase the hardware reliability models, software reliability models and open source software reliability models are discussed. A systematic survey of existing reliability models used in the industry for hardware and software both closed source as well as open source software, was attempted. Then a trial to explore techniques and technologies for measuring and improving software reliability and a framework is proposed to enable the early prediction of software reliability incorporating reliability measurement in each stage of the software development. The second phase involves the study and analysis of the role of Free and Open Source Software (FOSS) in different communities. A comprehensive survey has been conducted for studying why individual programmers, the government and many of the IT firms are lining up towards FOSS and the benefits by adopting FOSS for building their custom products have been studied. The analysis of the results are presented. This will be useful to the project managers for taking a decision for the adoption of FOSS. These can identify the constraints and in adopting the FOSS in the existing environment of an organization. The third phase is to study and evaluate existing FOSS and to arrive at a reliability growth model. The next phase is to develop an algorithm and a model for the estimation of computational reliability by incorporating both software and hardware components. The generated model is then compared with the theoretical and existing models; it has been and concluded that the model developed is a reliable one at the final phase. The entire process is detailed in chapter 3, the scheme of research work and methodology.

1.13 Outline of the Thesis

The rest of the thesis is organized as follows

Chapter 2 is a systematic survey of existing reliability models used in the industry for hardware and software both closed source as well as open source software. Then a trial to explore techniques and technologies for measuring and improving software reliability and a frame work is proposed to enable the early prediction of software reliability, incorporating reliability measurement in each stage of the software development.

Chapter 3 discusses the scheme of the research work and the methodology which involves studying the effects of failure of an actual software package and working towards formulating a reliability model taking into consideration the hardware issues.

Chapter 4 elaborates the study and analyses the role of Free and Open Source Software (FOSS) in different communities.

Chapter 5 focuses on the study and evaluation of existing OSS to arrive at a reliability growth model.

Chapter 6 formulates an algorithm for estimating software reliability and development of a simplified model.

Chapter 7 details the evaluation and comparison of the developed model with the application of a simplified model and its application in real time situation.

Chapter 8 recapitulates the thesis and provides conclusions and research findings. Some of the results have been published in international journals and in the proceedings of various national and international conferences.

Chapter 2

A Survey of Reliability Models

Contents

2.1 Introduction

Reliability modeling is the process of predicting or understanding the reliability of a component or system prior to its implementation. Two types of analyses that are often used to model a complete system availability behavior are, Fault Tree Analysis and Reliability Block diagrams. The Reliability growth models are categorized as hardware models and software models. Hardware Reliability Growth Models (HRGM) are generally categorized as probabilistic models and statistical models. In probabilistic reliability growth models – because of no unknown parameters associated with these models, the data obtained during the program cannot be incorporated (www.urel.feec.vutbr.cz/.../459.pdf). Statistical reliability growth models – unknown parameters are associated

with these models. In addition, these parameters are estimated throughout the development of the product in question.

In this chapter, we will be discussing in detail the software reliability growth models, hardware reliability growth models and open source software reliability models. And finally a frame work is proposed to enable the early prediction of software reliability as well as techniques and technologies for measuring and improving it.

2.2 Reliability Growth Models

A Reliability growth model provides a systematic way of assessing and predicting system reliability based on certain assumptions about the fault in the system in a usage environment. It involves comparing measured reliability at a number of points of time, with known functions that show possible changes in reliability. A reliability growth model is a model of how the system reliability changes over time during the testing process. As system failures are discovered, the underlying faults causing these failures are repaired so that the reliability of the system should improve during system testing and debugging. To predict reliability, the conceptual reliability growth model must then be translated into a mathematical model. Reliability growth models can therefore be used to support project planning.

A variety of models are available for the estimation of reliability in the case of software as well as hardware. The role that software plays as a support to modern societal activities cannot be underestimated. However,

the ability to predict software reliability is still not well understood and it needs further study. Although a number of software reliability models have been developed till date, none has been universally accepted in the field (Smidts and Li [2002], Li and Smidts [2003]).

A taxonomy of reliability models is as shown in the Fig.2.1. The figure shows present hardware, closed source software and open source software reliability models. The hardware reliability models include Weibull model, Constant hazard model and Linearly increasing model. The closed source software models are generally classified as failure rate models and Non Homogeneous Poisson Process (NHPP) models. The failure rate models are again divided as general and bayesian models. There are assessment and predictive models in the general category. In the case of OSS there are certain studies which concludes that the Weibull distribution can be used as a model.

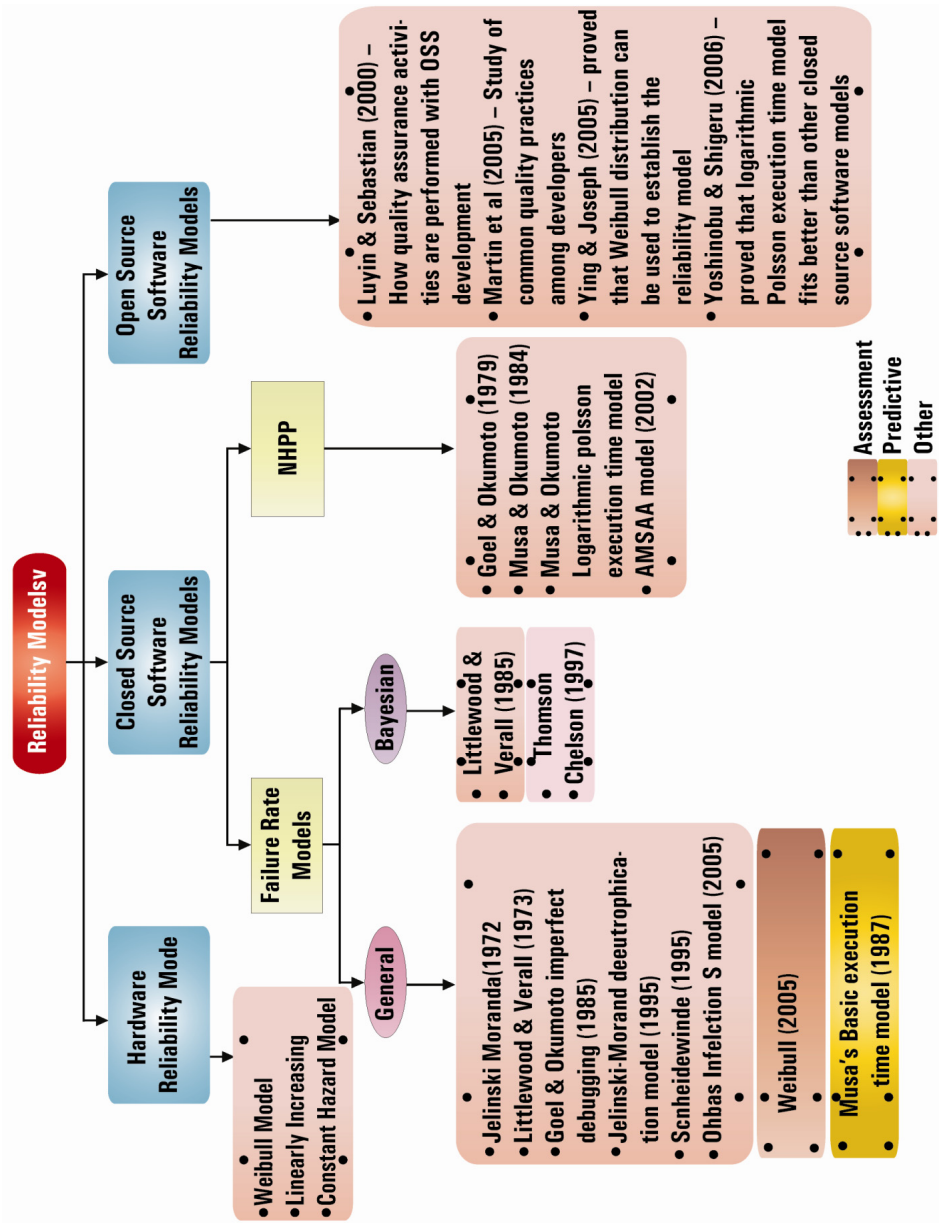


Figure 2.1 Taxonomy of software reliability models

2.2.1 Software Reliability Growth Models

Software Reliability is considered as part of software quality assurance and have many attributes including usability, capability, performance, functionality, documentation, maintainability and reliability. It is essentially being able to deliver usability of the services while assuring the constraints of the system. Software reliability modeling surprisingly to many, has been around since the early 1970s, with pioneering works by (Moranda[1972], Moranda[1975], Shooman[1972], Shooman[1973], Shooman[1976], Shooman[1977], Coutinho [1973]). The basic approach is to model past failure data to predict future behavior. The models fall into two basic classes namely failures per time period and time between failures.

A software reliability growth model provides a systematic way of assessing and predicting software reliability based on certain assumptions about the fault in the software and fault exposure in a given usage environment (Joe et. al [1993]). The reliability growth for software is the positive improvement of software reliability over time, accomplished through the systematic removal of software faults. The rate at which the reliability grows depends on how fast faults can be uncovered and removed. A software reliability growth model allows project management to track the progress of the software's reliability through statistical inference, and to make projections of future milestones (Lakey [1997], Musa and Okumoto [1983]). Models are classified in terms of five different attributes. Time domain: Wall clock

versus Execution time. Category: Total number of failures that can be experienced in finite or infinite time. Class/Finite failure category: Functional form of the failure intensity expressed in terms of time. Family/Infinite failure category: Functional form of the failure intensity function expressed in terms of the expected number of failures experienced. Type: The distribution of the number of the failures experienced by time t. Poisson and Binomial are the two important types.

A systematic frame work designed to predict software reliability from software engineering measures was summarized as follows (Li and Smidts [2000], Li and Smidts [2003], Smidts and Li [2000]). Research activities in software reliability engineering have been conducted over the past two decades and many Software Reliability Growth Models (SRGMs) have been proposed for the estimation of software reliability and number of faults remaining in the software (Goel and Okumoto [1979] , Hossain and Dhahiya [1993], Leung [1992], Ohba[1984], Pham [1993], Yamada [1985]). Most of the SRGMs assume that each time a failure occurs, the error which caused it, is immediately removed and no new errors are introduced (Hoang Pham and Xuemei Zhang [1999]).

Most models make some assumptions about the software failure process so that the model becomes mathematically tractable (Pankaj and Rajib [2011], Goel [1985]) has given the typical assumptions made in Software Reliability Model with its limitations. Reliability models have been proposed by Goel and Okumoto [1979], Jelinski and Moranda

[1972] , Littlewood and Verrall [1973] Musa and Okumoto [1984], and Shooman [1972]. To employ a model for reliability prediction, value of some of the parameters need to be specified. These are typically determined by analyzing the past failure data of the software.

The J-M model proposed by Jelinski and Moranda [1972] is one of the simplest and earliest of the software reliability models. The J-M model assumes that times between failures are independent random variables following exponential distributions, there are finite number of faults at the beginning of the test phase, and that the failure rate is uniform between successive failures and is proportional to the current error content(number of faults remaining) of the program being tested. This model is very simple to use. It is also fairly accurate for some data sets, but sometimes leads to inaccurate predictions. (Pankaj and Rajib Ghosh [2011]).

The basic execution model proposed by Musa et. al.[1987] make assumptions similar to the above model except that the process modeled is the number of failures in specified execution time intervals. There are a finite number of faults in the beginning of the test phase, and the times between failures are exponential, the failure rate being uniform between successive failures. He also provides a systematic approach for converting the model so that it can be applicable for the calendar time as well.

The Littlewood and Verrall model [1973] assumes exponential distribution for the random variable representing the failure interval time.

But the failure intensity is regarded as a stochastically decreasing function with gamma distribution, implying that the fault fixing process is not considered as perfect, and that faults are of different sizes. A user-controlled function determines the nature of the reliability growth. This model requires complex statistical inference for determining the parameters.

The Goel and Okumoto (G-O) model [1979] considers the software failure process as a Non Homogeneous Poisson Process (NHPP) with a mean function $\mu(t)$. This model treats initial error contents as a random variable.

The M-O model proposed by Musa and Okumoto [1984] views failure process as an NHPP like G-O model. But Unlike G-O model it assumes reduction in failure rates are greater for the earlier fixes. MO model assumes failure rate to be an exponential function of the expected number of failures. Input to the model is in the form $t_1,t_2,\ldots\ldots$,where each t_j represents the execution time. Execution time is related to calendar time through some suitable assumption and further computation.

As we can see, the basic input to all these models is the times of past failures, or times between consecutive failures. These data are used to determine the value of parameters, and then to predict the reliability of the given software. Most of the models use calendar time, and where execution time is used, suitable methods are used to convert it to calendar time.

Goel and Okumoto proposed an imperfect debugging model called Goel and Okumoto Imperfect Debugging Model (Amrit and Goel [1985]), which assumes that faults are removed with certainty when detected, is not always the case. In this model, the number of faults in the system at time t, X(t), is treated as a Markov process whose transition probabilities are governed by the probability of imperfect debugging. Times between the transitions of X(t) are taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the $(i-1)^{st}$ and ith failures is given by

$$Z(ti) = [N - p(i-i)]\lambda.$$

where N is the initial fault content of the system, p is the probability of imperfect debugging, and λ is the failure rate per fault.

Littlewood/Verrall Bayesian Model took a different approach to the development of a model for times between failures (Amrit and Goel [1985]). The times between failures are assumed to follow an exponential distribution but the parameter of this distribution is treated as a random variable with a gama distribution, viz.

$$f(ti/\lambda i) = \lambda i\, e^{-\lambda i\, ti}. \text{ And}$$

$$f(\lambda i/\alpha, \Psi(i)) = [\Psi(i)]^{\alpha}\ \lambda i^{\alpha-1}\ e^{-\Psi(i)}\ \lambda i / \Gamma(\alpha).$$

where $\Psi(i)$ represents the quality of the programmer, and the difficulty of the programming task. It is claimed that the failure phenomena in

different environments can be explained by this model by taking different forms for the parameter Ψ(i). This is a software reliability growth model based on stochastic differential equations for the integration testing phase of distributed development environment (http://www.coverity.com). This model has a simple structure, hence it is easily applied. This is very useful for software developers in distributed development environment in terms of practical reliability assessment.

Jelinnski - Moranda de-eutrophication model is an exponential Failure Time Class Model (Michael [1995]). The de-eutrophication model, developed by Jelinnski and Moranda, is still being applied today. The elapsed time between failures is taken to follow an exponential distribution with a parameter that is proportional to the number of remaining faults in the software, ie. The mean time between failures at time t is $1/\phi(N-(i-1))$. Here t is any point in time between the occurrence of the $(i-1)^{st}$ and the i^{th} fault occurrence. The quantity ϕ is the proportionality constant, and N is the total number of faults in the software from the initial point in time at which the software is observed. This is a binomial type model as per Musa and Okumoto's classification.

The Schneidewind's model is based on the fact that the current fault rate might be a better predicator of the future behavior than the observed rates in the distant past (Michael [1995]). The failure rate process may be changing over time and there are three forms of the model. Model 1: utilizes all of the fault counts from the n periods. This

reflects the view that all of the data points are of equal importance. Model 2: ignores the fault counts completely from the first through the s-1 time periods. ie. Use only the data from period s through n. This reflects the view that the early time periods contribute little if anything, in predicting future behavior. Model 3 is an approach intermediate between the first two, which reflects the belief that a combination of the first s-1 period is indicative of the failure rate process during the later stages.

The Geometric model is an infinite failure category model, and this is a variation from the Jelinski-Moranda model and was proposed by Moranda (Michael[1995]). The time between failures is taken to be an exponential distribution, whose mean decreases in a geometric fashion. The discovery of the earlier faults is taken to have a larger impact on reducing the hazard rate than the later ones. As failures occur, the hazard rate decreases in a geometric progression. The function is initially a constant, D, but it decreases geometrically ($0<\phi<1$), as each failure occurs. The change in the reduction of the function is seen to get smaller as more failures occur, reflecting the smaller impact of the later-occurring faults.

Thomson and Chelson Model is a Bayesian Model category (Lakey et. al. [1997]). The hazard function for this model is defined as $(fi+f_0+I)/(T_i+T_0)$. Where, fi is the number of failures detected in each interval, and T_i is the length of testing time for each interval i.

Musa Execution Time Model assumes that there are N software faults at the start of testing, each is independent of others, and is equally likely to cause a failure during testing. A detected fault is removed with certainty in a negligible time, and no new faults are introduced during the debugging process. The process modeled is the number of failures in specified execution time intervals (Amrit and Goel [1985]). The failure rate, or the hazard function for this model is given by

$$Z(\tau) = \phi f(N\text{-}nc)$$

Where, τ is the execution time utilized in executing the program up to the present, ϕ is a proportionality constant, which is a fault exposure ratio that relates fault exposure frequency to the linear execution frequency, f is the linear execution frequency, N is the initial fault content of the system and nc is the number of faults corrected during $(0, \tau)$. One of the main features of this model is that it explicitly emphasizes the dependence of the hazard function on execution time. Musa also provides a systematic approach for converting the model so that it can be applicable for calendar time as well.

Ohbas Inflection S Model is fairly a general model (Ying and Joseph [2005]). It allows forecasts to be made early in the test stage with percentiles that take into account the subjective judgment of the engineer with accuracy. The mean value function for Ohba's model is -

$$m(t) = N (1 – e^{-\Phi t}/1+\varphi e^{-\Phi t)}$$

where,

N = total number of failures that would occur in infinite time

Φ = failure detection rate

φ = inflection parameter

Musa-Okumoto Logarithmic Poisson Execution Time Model is similar to Goel Okumoto model, with the number of failures by some time τ is assumed to be a NHPP(non homogeneous poison process) with a mean value function.

Weibull distribution family is the most widely used lifetime distribution model (Ying and Joseph [2005]). The 2-parameter Weibull distribution has long been used to model reliability patterns due to its ability in describing failure modes like initial, random and wear-out.

The Weibull two-parameter, cumulative distribution function (CDF): (Richard and Ray [2002]).

$F(t) = 1- e^{-(t/\eta)\beta}$

Where F(t) = fraction of parts failing

t = failure time

η = characteristic life or scale parameter (MTTF)

β = slope or shape parameter

e = pi or 2.718281828

AMSAA Model is a NHPP model represented as: (Richard and Ray [2002]).

$$\rho(t) = \lambda\beta^{t\ \beta-1}, t>0, \lambda>0, \beta>0$$

Where, the $\rho(t)$ is an intensity function (ie. The probability of a system failure in the interval $(t, t+\Delta t)$.

With the discussion of different models, an appropriateness of model usage can be summarized. Based on the failure intensity vs cumulative failures increasing, decreasing or a combination, we can suggest the appropriate models (Lakey et. al. [1997]). If it is increasing, the S-shaped and Weibull models can be used. If decreasing and the software has been in operation for some time without a failure, the Thompson Chelson Model can be used. Further classification can be done from the historical or collected data such as initial failure rate, estimated number of inherent faults, or the expected rate of the failure intensity. From the initial failure rate data, Musa Logarithmic model can be applied, Goel Okumoto model and Musa Logarithmic model can be applied on the inherent faults, and rate of change of failure intensity data collected as shown in the Fig. 2.2 (Lakey et. al [1997]).

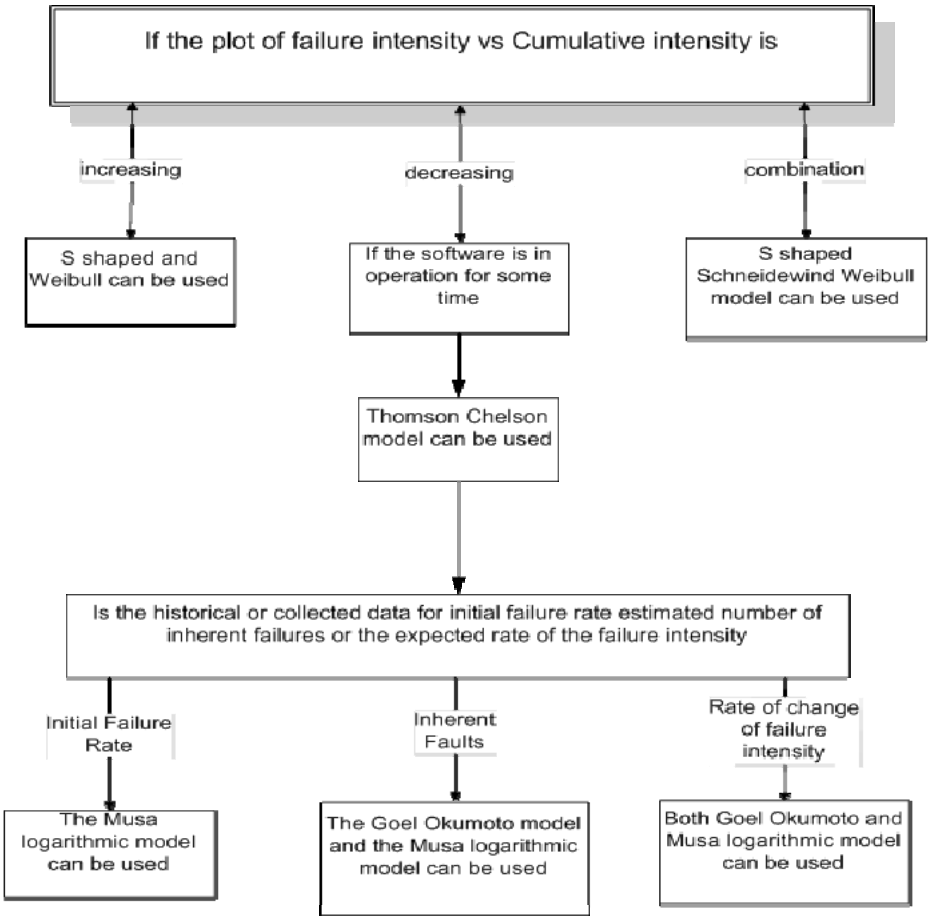


Figure 2.2 Appropriateness of Software Reliability Growth Models

2.2.2 Hardware Reliability Growth Models

A hardware reliability growth model is used to mention product reliability in the period during which, the observed reliability advances towards the inherent reliability of the product. Hardware and software reliability predictions adjusted by their respective growth models to coincide with the same point in time can be combined to obtain a

prediction of the overall system reliability. There were a number of reliability growth models suggested for hardware reliability in the literature. In this section, we outline important models that discuss hardware reliability. Understanding the dynamic behavior of system reliability becomes an important issue in either scheduling the maintenance activities or dealing with the improvement in the revised system design. In doing so, the failure or hazard rate function should be addressed. Bathtub curve is usually adopted to represent the general trend of hazard rate function as shown in Fig. 2.3. This curve exhibits three distinct zones. The first is, the short initial period called variously the early failure, infant mortality, or the burn in period. The decreasing but greater failure rate early in the life of the system is due to one or more of several potential causes. The causes include inadequate testing or screening of components during selection or acceptance, damage to components during production, assembly, or testing, and choice of components which have too great a failure variability. It shall be a specific goal of the supplier to ensure that the early failure period is rigorously controlled and covered by a suitable warranty (Shooman [1968], Thomas [1973].

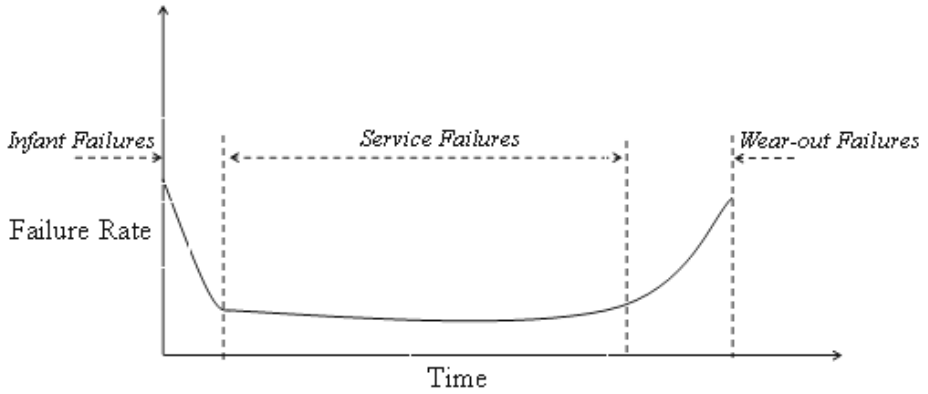


Figure 2.3 Bath Tub Curve for Hardware

The failures in the second zone are termed service failures. During this period, the failure or hazard rate is constant and it represents the effective life of the product.

The failures in the third zone are the wear-out failures. The incidence of failure in this zone is high since most of the components will have exceeded their service life, and consequently would have deteriorated. Hence, they are appropriately called wear-out failures.

Many studies were concentrated on depicting the geometric shape of the bathtub curve. The early contributors in this area include , Bain [1974], Smith &Bain [1975], Gaver [1979], Hijroth [1980], Dhillon [1981], Lawless [1982], Jaisingh et. al. [1987], Haupt & Schabe [1992], Schabe [1994], Xie and Lai [1996], Edelstein [1998]. Wang et. al. [2002]) proposed a general form of bathtub shape hazard function in terms of reliability. The relation between hazard rate and reliability of a system follows the definition (Wang et. al. [2002]).

$$Z(t) = -\frac{1}{R(t)}\frac{dR(t)}{dt} \quad \text{---------- (2.1)}$$

Usually the reliability decreases monotonically with time and thus there is a one to one correspondence between reliability and time. That is, the hazard rate function can also be expressed as

$$Z(t) = -\frac{1}{R(t)}\frac{1}{dt/dR(t)} = Z(R) \quad \text{---------- (2.2)}$$

Thus, instead of the usual procedure of estimating Z(t) the relationship of Z(R) based on the available data was defined. The change of expression Z (t) to Z(R) has certain advantages. First, the equation of dynamic reliability takes an autonomous form; particularly it belongs to a general type of logistic equation encountered very often in ecological science (Edelstein [1988]). Therefore good experience can be guided from these studies. Secondly, the hazard rate is investigated in finite domain (1, 0) as compared with that in infinite domain of time sequence.

Wang et. al. [1993] developed reliability models that can be applied for the development of a new mechanical product with modified function requirements. Wang et. al. [1996] also developed reliability models for material fracture due to crack growth.

The data obtained from failure tests can be analyzed to obtain reliability, failure density, hazard rate and other necessary information (Srinath [1991]). Obviously, the behavioral characteristics exhibited by one class of components differ from those exhibited by another class of

components. In order to compare different behavioral characteristics and also to draw general conclusions from behavioral patterns of similar components, a mathematical model representing the failure characteristics of the components becomes necessary. The procedure involves assuming a function for hazard rate, and thereby obtaining reliability and failure density by using this failure rate function. The assumed function for the hazard rate will be the hazard model. Some of the common hazard models are discussed below:

One of the most commonly used models is the constant hazard model. Here the failure rate is assumed to remain constant with time. That is, $Z(t) = \lambda$, a constant (Musa [2005]).

$$R(t) = \exp\left\{-\int_0^t Z(\xi)d\xi\right\} = \exp\left\{-\int_0^t \lambda d\xi\right\} = \exp\left\{\left[-\lambda\xi\right]_0^t\right\} = \exp(-\lambda t)$$

That is, for a constant hazard model, Reliability, $R(t) = e^{-\lambda t}$

Probability of failure, $F(t) = 1 - R(t) = 1 - e^{-\lambda t}$

Failure density, $f_d(t) = Z(t)R(t) = \lambda e^{-\lambda t}$

The variation of failure rate, reliability, probability of failure, and failure density with respect to time for a constant hazard model is shown in the following figure Fig. 2.4 (Srinath [1991]).

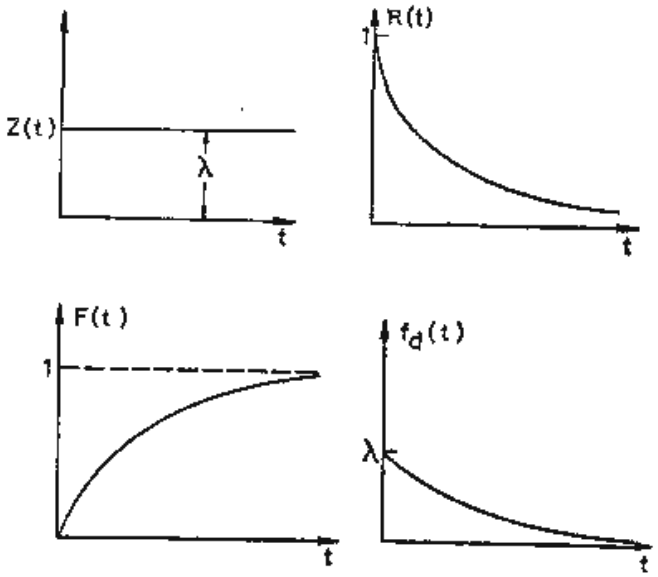


Figure 2.4 Variation of Failure Rate, Reliability, Probability of Failure, and Failure Density for a Constant Hazard Model (Srinath [1991])

It can be seen that, for a constant hazard model the mean time to failure is the reciprocal of failure rate.

That is,

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \left[-\frac{e^{-\lambda t}}{\lambda}\right]_0^\infty = -\frac{1}{\lambda}\left\{e^{-\infty} - e^{0}\right\} = -\frac{1}{\lambda}(0-1) = \frac{1}{\lambda}$$

The constant hazard model is also known as exponential reliability case.

In the case of linearly increasing hazard model the hazard rate is assumed to increase linearly with time. That is, $Z(t) = Kt$, where K is a constant

$$R(t) = \exp\left\{-\int_0^t Z(\xi)d\xi\right\} = \exp\left\{-\int_0^t K\xi d\xi\right\} = \exp\left\{\left[-\frac{K\xi^2}{2}\right]_0^t\right\} = \exp\left(-\frac{Kt^2}{2}\right)$$

That is, for a linearly increasing hazard model, Reliability, $R(t) = e^{-\frac{Kt^2}{2}}$

Probability of failure, $F(t) = 1 - R(t) = 1 - e^{\frac{-Kt^2}{2}}$

Failure density, $f_d(t) = Z(t)R(t) = Kt\ e^{-\frac{Kt^2}{2}}$

The variation of failure rate, reliability, probability of failure, and failure density with respect to time for a linearly increasing hazard model is shown in the following figure Fig. 2.5 (Srinath [1991]).

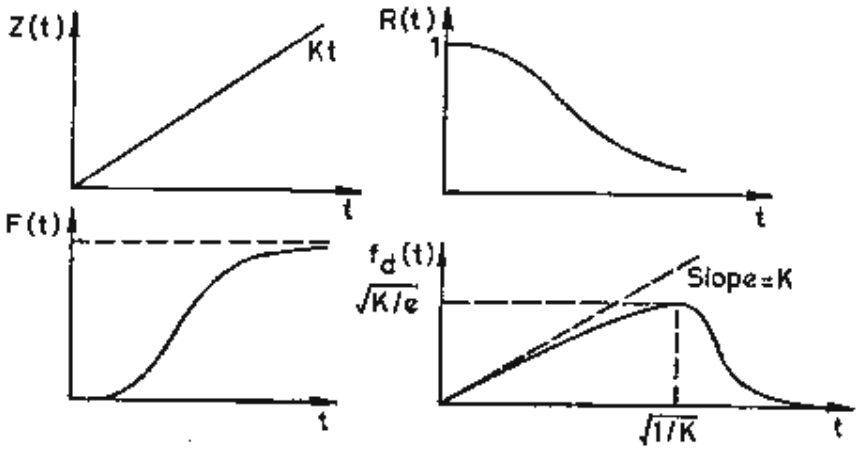


Figure 2.5 Variation of Failure Rate, Reliability, Probability of Failure, and Failure Density for a Linearly Increasing Hazard Model (Srinath [1991])

It can be seen from the failure density curve that the curve has a slope equal to K at time $t = 0$. Also the value of $f_d(t)$ reaches a maximum of $\sqrt{\frac{K}{e}}$ at time $t = \sqrt{\frac{1}{K}}$, and tends to zero as *t* becomes larger.

Another very popular model is the Weibull Model (Srinath [1991]) and is expressed as $Z(t) = Kt^m, m > -1$

Here *K* and *m* are parameters and if these are chosen appropriately, a variety of failure-rate situations can be covered, including both the constant hazard and linearly increasing hazard conditions.

If $m = 0$; $Z(t) = K$ and refers to a constant hazard model

If $m = 1$; $Z(t) = Kt$ and refers to a Linearly increasing model

The reliability can be expressed as

$$R(t) = \exp\left\{-\int_0^t Z(\xi)d\xi\right\} = \exp\left\{-\int_0^t K\xi^m d\xi\right\} = \exp\left\{\left[-\frac{K\xi^{m+1}}{m+1}\right]_0^t\right\} = \exp\left(-\frac{Kt^{m+1}}{m+1}\right)$$

That is, in case of Weibull model, Reliability, $R(t) = e^{-\frac{Kt^{m+1}}{m+1}}$

Probability of failure, $F(t) = 1 - R(t) = 1 - e^{\frac{-Kt^{m+1}}{m+1}}$

Failure density, $f_d(t) = Z(t)R(t) = Kt^m\, e^{-\frac{Kt^{m+1}}{m+1}}$

Following figure Fig. 2.6 shows the variation of reliability in case of Weibull model for various values of *K* and *m* (Srinath [1991]).

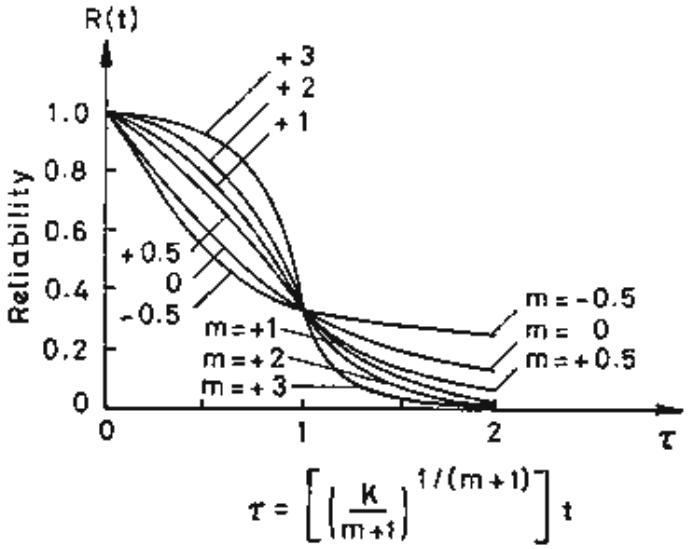


Figure 2.6 Variation of Reliability in case of Weibull Model

2.2.3 Reliability Models for Open Source Software

The OSS development mainly depends on the practice of welcoming every enthusiastic individual who would like to contribute to the project. On top of this, the freedom of using, modifying and distributing OSS leads to more robust software and more diverse business models (Wu and Lin [2001]). Software reliability models are useful to assess the reliability for quality management and testing progress control of software development. Although open source practices have been remarkably successful in recent years, the open source development model faces a number of product quality challenges. Rare open source projects have been archived successfully as a high level quality end product. However, these mature and successful projects face quality problems too. Even though lots of models and tools have

been suggested for reliability checking, very few models are applied and tested in this case.

One of the recent studies by Coverity Inc [http://www.coverity.com] on measuring reliability of open source software claims that the LAMP stack – Linux, Apache, MySQL, and Perl/PHP/Python – showed significantly better software quality above the baseline with an average of 0.290 defects per thousand lines of code, compared to an average of 0.434 for the 32 open source software projects analyzed. One of their goals of research on software quality, was to define a baseline so that people can measure software reliability in both open source and proprietary software projects.

Luyin and Sebastian [2000] discussed how quality assurance activities are performed within the OSS development. They pointed out that OSS development is very different from the traditional software development used in most of the software industry. Moreover, the quality assurance activities are also performed in a different fashion.

Martin et.al. [2005] have done exploratory interviews with free and open source developers to study the common quality practices among the developers to implement a quality process improvement strategy. They found that even though development of OSS projects share common practices the quality of the resulting products needs further empirical evaluation. This implies that we have to look into reliability models for open source software development.

An empirical study towards open source software reliability model was conducted by Ying and Joseph [2005]. They have collected data from eight active open source projects from "SourceForge.net" and reliability analysis was done based on the bug arrival rate. They claim that general Weibull distribution is a possible way to establish the reliability model. Further, in contrast with closed source projects, it is unlikely to find a Rayleigh curve, to model all open source projects.

In a recent study on Xface desktop environment, an open source distributed project, Yoshinobu and Shigeru [2006], attempted an evaluation under Mozilla public license by applying various reliability growth models. Conventional models like exponential growth model, delayed S-shaped model, inflection S-shaped model and logarithmic Poisson execution time model were considered and goodness-of-fit comparison were done. Various software reliability assessment measures were derived from the non-homogeneous Poisson process (NHPP) models. It has been concluded that the logarithmic Poisson execution time model fits better than the other software reliability growth models for the actual data set.

2.3 Computational System Reliability

Computational system reliability is concerned with hardware reliability, software reliability, reliability of interaction between hardware and software and reliability of interaction between the system and the operator. In general, a system may be required to perform various functions, each of which may have a different reliability. In addition, at different times, the system may have a different probability of successfully

performing the required function under stated conditions. The analysis of the reliability of a system must be based on precisely defined concepts.

Software intensive systems are increasingly used to support critical business and industrial processes, such as in business information systems, e-business applications, or industrial control systems. Reliability engineering gains its importance in the development process. Reliability is compromised by faults in the system and its execution environment, which can lead to different kinds of failures during service execution: Software failures occur due to faults in the implementation of software components, hardware failures result from unreliable hardware resources, and network failures are caused by message loss or problems during inter component communication (Franz and Heiko [2012]).

The analysis of the reliability of a system must be based on precisely defined concepts. Since it is readily accepted that a population of supposedly identical systems, operating under similar conditions, fall at different points in time, then a failure phenomenon can only be described in probabilistic terms. Thus, the fundamental definitions of reliability must depend on concepts from probability theory (Pham [2007]).

System-level reliability and availability requirements set forth by U.S. Government agencies procuring large software intensive systems encompass both hardware and software. However, specifications, statement of work requirements, and compliance documents (standards) usually implicitly or explicitly focus on hardware and are largely silent about software reliability, maintainability, availability and dependability.

Consequently, contractor system reliability analyses and design reviews usually ignore quantitative software reliability, maintainability, availability, and dependability requirements. During system testing and evaluation, data on software operating times, failure rates, and recovery times are not collected. Finally, logistics and support specialists devote significant attention to sparing and maintenance concept development, but often do not adequately consider the software-related sustainment issues of large computer systems. These problems can be solved, by an appropriate definition of requirements for software-intensive system reliability in specifications, and in the definition of programmatic requirements in contractual documentation (Myron et. al.[2007]).

In general, a system may be required to perform various functions, each of which may have a different reliability. In addition, at different times, the system may have a different probability of successfully performing the required function under stated conditions. The term failure means that the system is not capable of performing a function when required. The term *capable* used here is to define if the system is capable of performing the required function. However, the term *capable* is unclear and only various degrees of capability can be defined (Musa [1980], Pham [2007]).

2.4 A Framework to Enable the Early Prediction of Software Reliability

The objective is to develop a framework to enable the early prediction of software reliability incorporating reliability measurement in each stage of the software development. Leslie et.al.[2008] state that

the ability to predict the reliability of a software system early in its development can help to improve the systems quality in a cost effective manner. Therefore, the proposed framework, measures and minimizes the complexity of software design at the early stage of software development lifecycle, leading to a reliable end product. To calculate the reliability of software product, the reliabilities at different stages of product development like requirements analysis, design, development, testing and implementation etc. will have to be evaluated. This facilitates the improving of the overall product reliability. It is observed that modifications and error identifications during operation and implementation can lead to re-engineering of large parts of the system, which has been shown to be costly. Hence to ensure the quality of the developed system, it is important to ensure quality at different stages of development. A few approaches which do consider component-level reliability (Goseva et. al. [2003], Reussner et, al. [2003]) , assume that the reliabilities of a given component's elements, such as its services, are known.

Reliability prediction is useful in a number of ways. A prediction methodology provides a uniform, reproducible basis for evaluating potential reliability during the early stages of a project. Predictions assist in evaluating the feasibility of proposed reliability requirements and provide a rational basis for design and allocation decisions.

2.4.1 Background

The attention of scientists and engineers in the late 60's was focused mainly on hardware reliability, mechanical and electronic

systems. Then from 70's onwards the permanent growth of software applications became the center of many studies. Computers are applied in almost all areas of human life. The main applications includes banking system, power distribution, hospital management and critical systems like air traffic control and airplane flight, where failure could lead to catastrophes and loss of many lives(Vladimir et. al. [2011]). On one hand there is increasing dependence on software and on the other hand, software systems are becoming more and more complex and harder to develop and maintain. Software functionality is becoming crucial from the aspects of reliability, safety of human lives and security issues as well (Vladimir et. al. [2011], Voas and Payne [2000]).

2.4.2 Reliability Prediction

Reliability predictions are conducted during the requirement and definition phase, the design and development phase, the operation and maintenance phase in order to evaluate, determine and improve the dependability measures of an item. Successful reliability prediction generally requires developing a reliability model of the system considering its structure. Several prediction methods include reliability block diagrams, fault tree analysis, state-space method etc.(www.epsma.org).

A prediction scenario is shown in the Fig 2.7. The method involves collection of failure data from the field and it is compared with the information available in the database. The database is updated regularly to keep it current. The failure rate figures are employed, tested and checked in some suitable reliability models to predict the reliability.

This will help the project managers to predict and develop a reliable product.

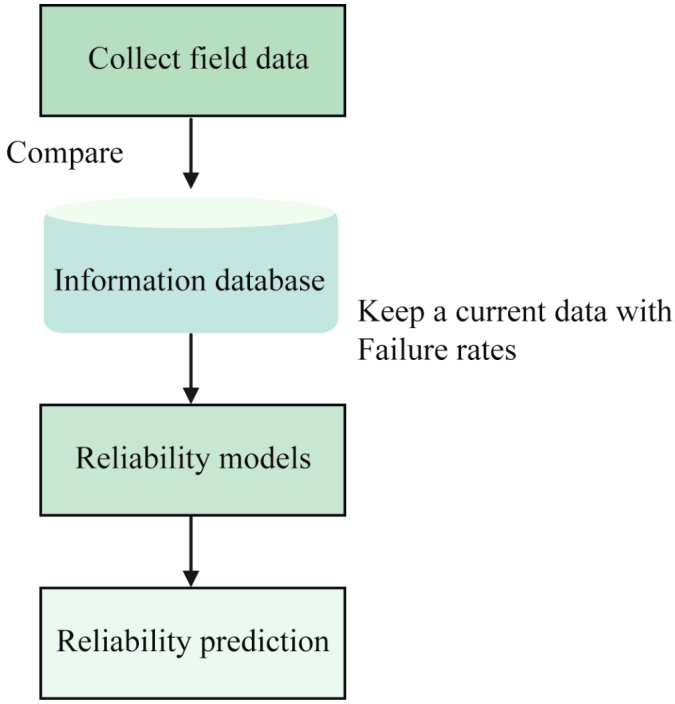


Figure 2.7 Prediction Method

2.4.3 The Framework

The main task of a system program office (SPO) when acquiring a new software system, is to specify the requirements to the developer and to see that the requirements mentioned are met as the system development process evolves to the final product. It is also necessary to assure that the qualities of the software such as reliability, maintainability, usability, testability, and portability are attained.

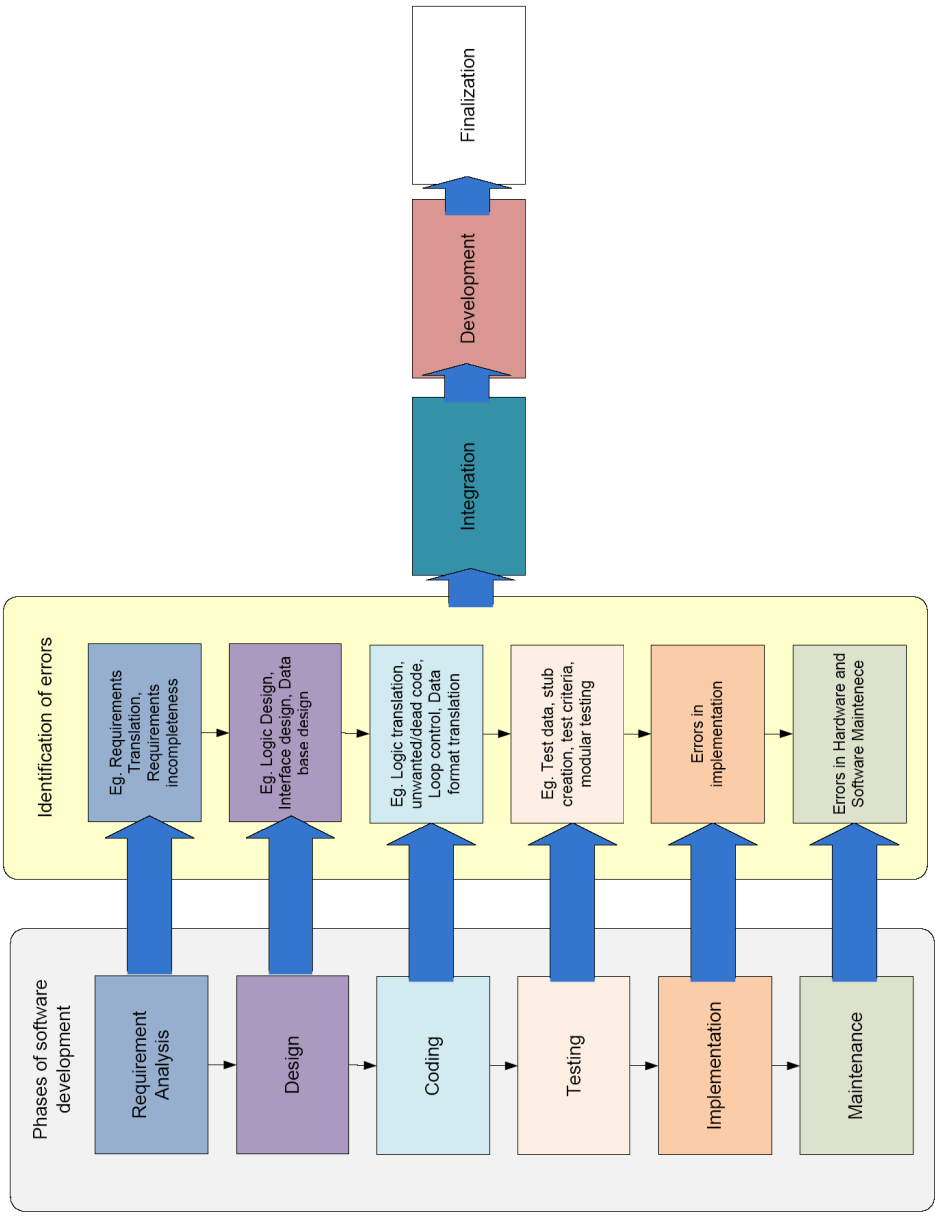


Figure 2.8 Detailed software reliability prediction frame work

An error in the software product can occur when there is a difference between the actual output of the software and the expected correct output. Fault is a condition that causes a system to fail to perform its required function. Failure occurs when the behavior of the software is different from the specified behavior (Amitabha and Khan [2012]).

A reliability prediction framework is shown in the Fig 2.8, which is to analyze the reliability at different stages of development. The process includes phases of software development, identification of errors, integration, development and finalization. The first step is to analyze the phases of development, which includes requirement analysis, design, coding, testing, implementation and maintenance. Next comes the identification of errors in different phases, where possible occurrences of errors are identified. The collected error data is used to calculate the failure density and thereby the reliability.

In the Identification phases, Software reliability attributes have been identified in different phases of development. Firstly, draw up a functional profile, then identify the needs of software reliability, then define the fault/failure type and the fault/failure severity, finally, understand the software development process and environment.

Integration phase relation between reliability aspects of the above identified phase is determined. The next stage is to formulate a plan to integrate the software aspects to incorporate reliability criteria in the software development stage.

Reliability estimation model (REM) is developed in the development phase. In this phase, first of all, establish a data collection plan and collect data through templates. Secondly, draw up an operational profile and allocate software reliability goal. Predict software reliability through software prediction models and estimate software reliability through software estimation models. Elicit improvements and review improvements and establish a device for software reliability improvement (Voas [2000]). Finally on the basis of the review, the whole approach is reviewed and revised if needed.

2.5 Techniques and Technologies for Measuring and Improving Software Reliability.

Reliability measurement is a set of mathematical techniques that can be used to estimate and predict the reliability behavior of software during its development and operation. The primary goal of software reliability modelling is to find out the probability that it will fail in a given time interval, or, what is the expected duration between successive failures (Allen and Lyu [1999], Allen and John [1998]). Software reliability is closely influenced by the creation, manifestation and impact of software faults. Consequently, software reliability can be improved by treating software faults properly, using techniques of fault tolerance, fault removal, and fault prediction. Fault tolerance techniques achieve the design for reliability, fault removal techniques achieve the testing for reliability, and fault prediction techniques achieve the evaluation for reliability (Lyu[1998]).

Reliability engineering is a daily practised technique in many engineering disciplines. Using a similar concept in these disciplines, we define *software reliability engineering* as the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability. Software reliability engineering therefore, includes (Lyu [1996]): (1) software reliability measurement, which includes estimation and prediction, with the help of software reliability models established in the literature; (2) the attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability; and (3) the application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance.

To achieve software reliability, different techniques for measurement have been developed. The main purpose is to test the software and measure the reliability according to the predefined criteria of the techniques. The result of this offers the developers and users an understanding of the reliability of the software (lyu [1996]). This process is known as reliability engineering and can be summarized as shown in the Fig 2.9.

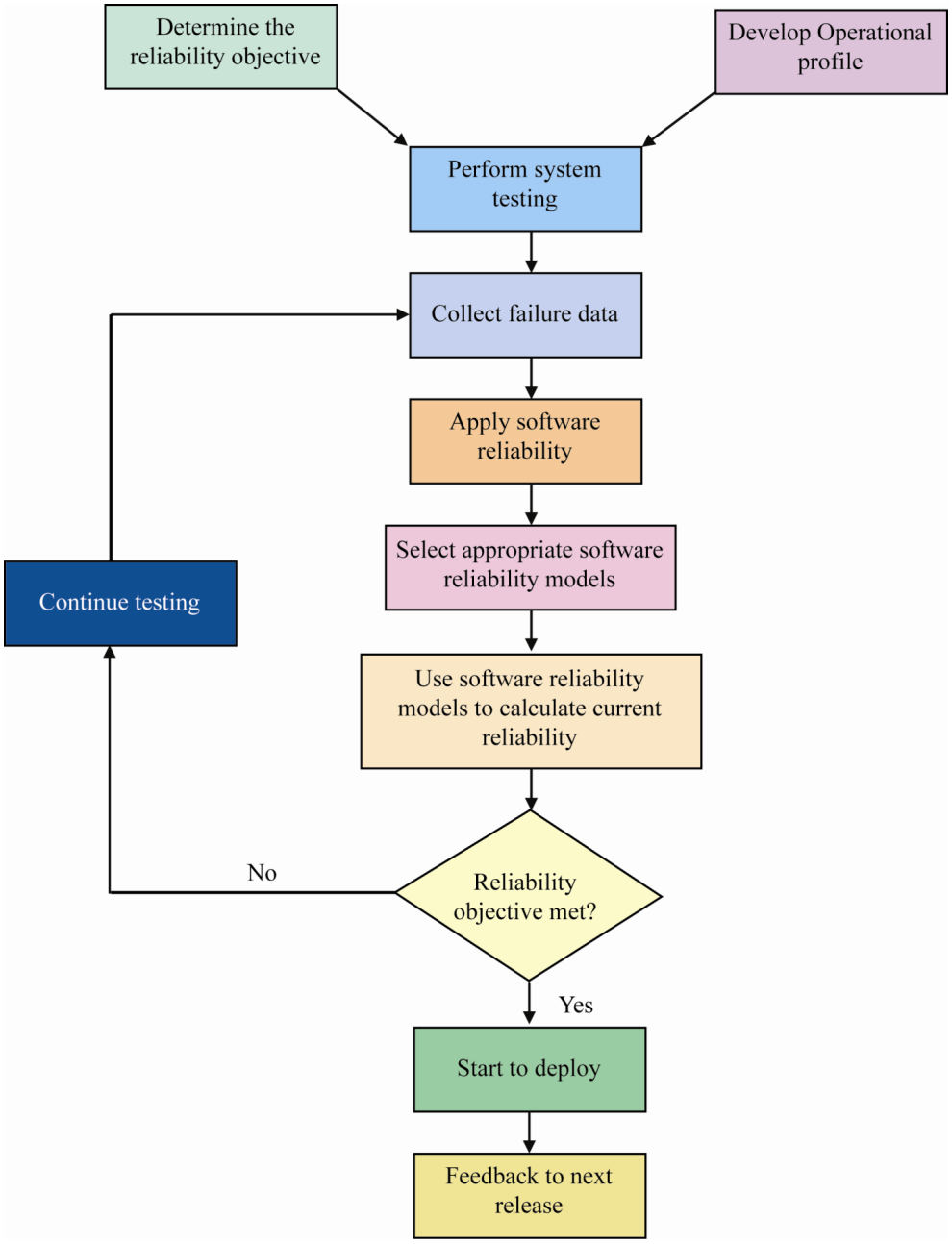


Figure 2.9 Software Reliability Engineering Process Overview (lyu [1996])

2.6 Conclusion

From the above discussions it is evident that even though a lot of models are developed and available in the literature for evaluation of software reliability, all the models do not provide a direct quantification of the reliability, that is, all these models are not necessarily deterministic. Typical hardware reliability models make use of the available component field failure data for reliability estimation. However no attempts were made to incorporate hardware and software together in reliability estimation and the present work is emphasized on this. Also in the early stages of development, failure information is not available to quantitatively measure reliability of a software product. Software reliability cannot be calculated during the requirement analysis, design, development, testing and maintenance phases, if adequate data on system failures is collected throughout the project. The same models for estimating reliability parameters, such as the expected number of failures in a certain period of time, failure intensity, the expected time of the next failure, etc., could be applied to software systems as well. A software reliability prediction framework is proposed, which enhances the reliability calculation at different stages of development and hence increases the end product reliability.

Chapter 3

Scheme of Research work and Methodology

Contents

3.1 Introduction

The methodology involves the study of the existing reliability models, the study and analysis of the role of free and open source software in different communities, the study and evaluation of existing open source software and to arrive at a reliability growth model, the development of a model for estimation of computational reliability by incorporating both hardware and software components, and to compare the actual reliability with the developed model and other existing models as shown in Fig 3.1.

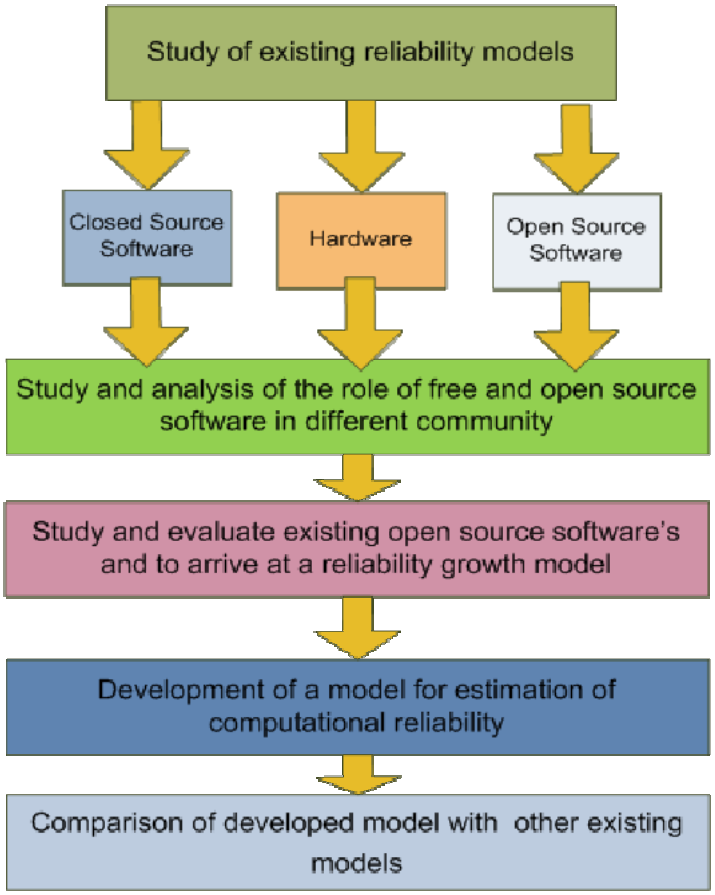


Figure 3.1 Design and Development Phases in the Study

Study of the existing reliability models discusses the software, hardware, open source reliability models and the appropriateness of model usage. Study and analysis of the role of free and open source software in different communities involves the preparation of a questionnaire and data collection. Recently software industry started adopting existing open source code, open source libraries etc in developing proprietary products. A case study has been done for

studying why individual programmers, Government and many of the IT firms are lining up towards FOSS and the benefits of adopting open source software for building their custom products. A comprehensive survey has been conducted among the above, to identify the role of these communities towards the open source software, and an analysis of the results are presented. The constraints to adopt the FOSS in the existing environment of an organization are also discussed.

Studying and evaluating of existing open source software, and arriving at a reliability growth model phase involves collection of bug tracking data from a few popular open source projects and the time related bug arrival investigation. It further involves a proposal of model for software development using open source software and discussion of problems associated with the integration of the open source with custom made software.

The development of a model for estimation of computational reliability involves studying the effects of failure of an actual software package and working towards formulating a reliability model taking into consideration the hardware issues.

3.2 The Methodology for Model Development

The methodology involves defining a system configuration consisting of software and hardware elements. The software and hardware components are considered independently consisting of two subsystems. The failure related data of hardware components are based

on the field data. On the other hand, the software failure is based on the bug arrival rate as published in the Debian site. The hardware modeling is based on the constant Hazard model whereas, the software model is based on the distribution as obtained from the bug arrival. A relatively simplified model is one, where both hardware and software exhibit a constant failure.

Software reliability models are used to assess a software product's reliability to estimate the number of latent defects when it is available to the customer (Leblanc and Roman [2002]). This estimation is important for two reasons namely to provide an objective statement of the quality of the product, and to prepare the resource plan for the software maintenance phase.

The criterion variable under study, is the number of defects or defect rate normalized to lines of code, or function points in specified time intervals or time between failures. Reliability models can be either static or dynamic. Static models use attributes of the project or program modules to estimate the number of defects in the software, whereas dynamic models, based on statistical distributions, use the current development defect pattern to estimate end-product reliability.

The objective of the research is to develop a model to represent reliability of a computing system by considering both hardware and software failure impacts. The methodology involves studying the effects of failure of an actual software package and working towards formulating a

reliability model, taking into consideration the hardware issues. Studying the effects of failure of actual software package is comprised of three stages namely data collection, data preprocessing and analysis. The formulation of the reliability model involves algorithm development, model development, and comparison of theoretical products with the formulated model as shown in Fig. 3.2. The first phase of the work is the data collection. In this phase, failure data for software and hardware are to be collected. The collected data is pre-processed to get the valid data set in the second phase of the work. In the third phase the valid data set is analyzed to get the reliability equation and thus the model generation. The generated model is then compared with the theoretical and existing models and concluded that the model developed is a reliable one as the final phase.

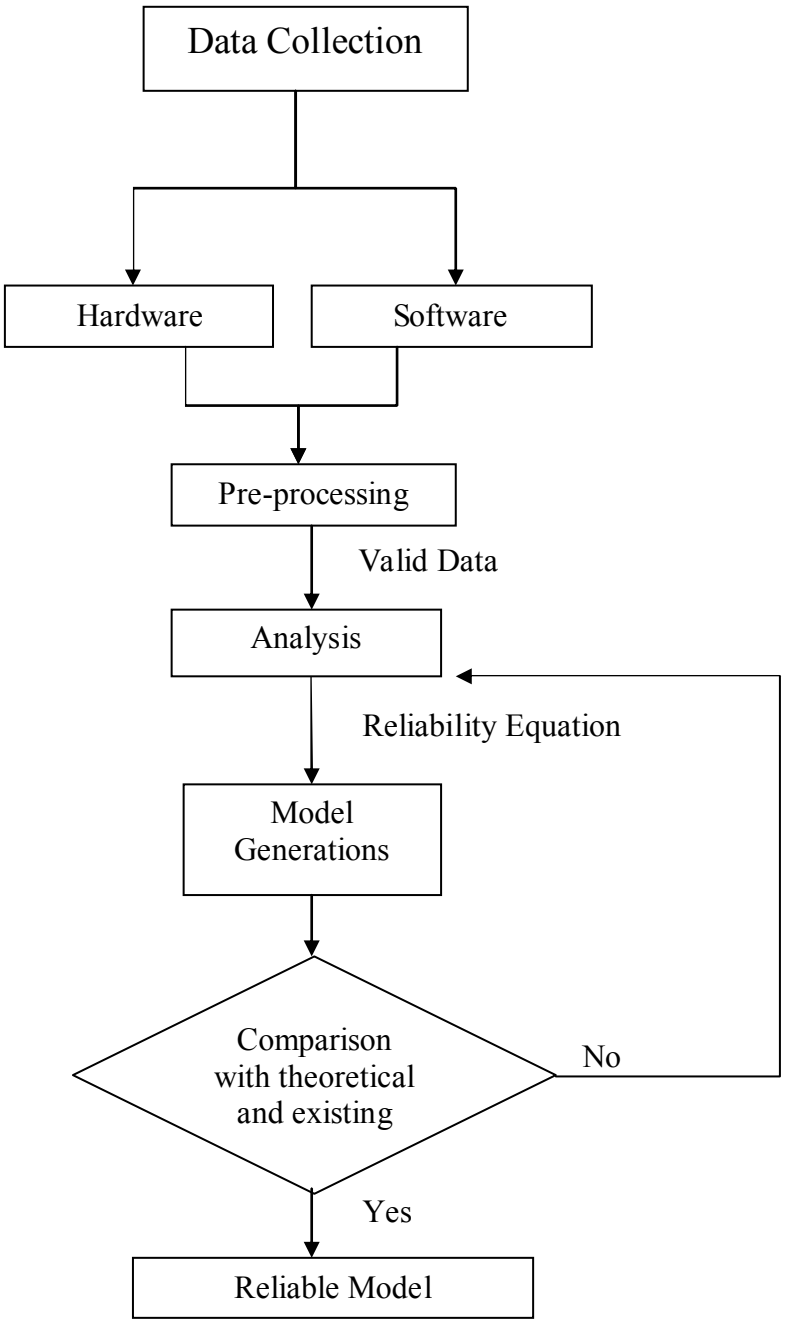


Figure 3.2 Methodology for Model Development

3.2.1 Analysis Phase

For reliability analysis of hardware and software package, information regarding its failure has to be studied. So, this phase can be further broken down into data collection and analysis. The former will deal with collecting parameters essential to evaluate its reliability, and the latter will deal with the evaluation and analysis.

3.2.2 Data Collection

The reliability of software is adversely affected by failures or bugs in computer programs. A failure is the departure of software behavior from user requirements. A static fault (or bug) in the software code causes failure as soon as it is activated during program execution. Hence as a first step towards building a reliability model, failure reports were collected to evaluate the reliability of a software package due to occurrences of bugs. The data required was the time of identification of a bug and time of repairing the same, so that the total failure duration could be obtained. From the failure duration the rate of failure could be calculated.

In the case of hardware, failure data are collected from the production system of Cochin University of Science and Technology (CUSAT) where Debian based server system were used to run their website, mail server etc. The failure data are collected systematically whenever a system failure had occurred. The collected data is analyzed to reach to the required refined set of data.

The bug reports were collected mainly from online open source development site Debian.org. Debian GNU /Linux is a free operating system that comprises of 25000 packages, precompiled in a user-friendly format. It is developed through distributed development all around the world. The Debian GNU/Linux distribution has a bug tracking system which consists of bugs reported by users and developers. This facility was used as the primary data source.

The bug report generated by the Debian's Bug Tracking System (BTS) has a typical format where each bug related to a package is assigned a unique bug identity. The bug status report gives the package name, bug identity, its description along with the author, and its present status. The following tags are used to indicate the present status of a bug. P: pending, +: patch, H: help, M: moreinfo, R: unreproducible, S: stable, U: upstream and I: squeeze-ignore.

The second set of tags indicate what releases a bug applies to: O for old stable (sarge), S for stable (lenny), T for testing (squeeze), U for unstable (sid) or E for experimental. The detailed operations carried out are detailed in the Appendix part of the thesis.

3.2.3 Data Preprocessing

Data preprocessing is an important step to refine the collected data. Generally the real world data is incomplete, lacking attribute values, lacking certain attributes of interest, or containing only aggregate data or may be Noisy: containing errors or outliers or can be Inconsistent: containing

discrepancies in codes or names. Data preprocessing includes data cleaning, data integration, data transformation, data reduction and data discretization.

Data cleaning usually includes fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies, whereas, data integration is carried out by using multiple databases, data cubes, or files. Data transformation is the normalization and aggregation process. Data reduction is reducing the volume but producing the same or similar analytical results and Data discretization is part of data reduction by replacing numerical attributes with nominal ones.

3.2.4 Data Analysis and Interpretation

The collected data is analyzed in order to arrive at reliability. The software bug arrival is plotted against time and the failure function is derived. This function is used for arriving at the software reliability. In the case of hardware the mean time to failure for each of the components is sufficient to arrive at the reliability based on the constant hazard model.

In the case of the software a total of 1880 packages were available at the start of the analysis, as per the details available from the official website of Debain [http://www.debian.org]. This is taken as the initial population. A time interval of one month is fixed and the bug arrival rate during this interval is noted. The observations are taken for 1 year after which the bug arrival is negligible indicating that the software has more or less stabilized.

3.3 Algorithm Development

A systematic procedure for evaluating reliability of open source software is developed by considering the prevailing trends in industry. The methodology involves defining an equation for the pattern of failure based on the available bug arrival rate and developing a generalized model for the reliability of the software.

3.4 Model Development

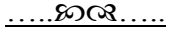
The reliability estimation involves considering the computing system as two subsystems, one comprising of hardware components and the other, the various software modules or packages. These various packages are considered as various components of the software part of the system [http://www.debian.org]. The software and hardware components are assumed to be connected in series and based on the reliability block diagram, the overall system reliability is obtained.

3.5 Comparison of Developed Model with Other Existing Models

The developed model arrives at the reliability by combining software and hardware parameters. A comparison is made with the other existing reliability models so as to arrive at the error involved in reliability analysis when computation of reliability is calculated without considering the software and hardware elements together.

3.6 Conclusion

The research work is aimed at arriving at a model for reliability by combining hardware and software reliabilities for FOSS. The methodology involves the analysis of software reliability by collecting bug reports of software packages and analyzing the failure rate thereby evaluating its reliability. The reliability of the hardware part is obtained using the collected component failure data by employing a constant hazard model. Finally, the hardware and software reliabilities are integrated to arrive at the overall system reliability.



Chapter 4

Role of Community and Open Source Software a Case Study

Contents

4.1 Introduction

Free and Open Source Software (FOSS) development has emerged as one of the more important Information Technology (IT) trends in this century. Recently, software industries including Government organizations adopted FOSS for building their software based infrastructure for various reasons. Today, individual programmers, Government and many of the IT firms are lining up towards FOSS and are adopting open source software for building their custom products.

Open Source software is becoming more and more popular and is achieving a high rate of growth. Open source code evolves through community cooperation. These communities are composed of individual programmers, very large companies for business purpose and the government. Even individuals can participate in open source software projects in a number of ways, such as contributing source code, testing and installing the software, reporting and fixing bugs, writing documentation and posting forum messages (Scott and Greg [2007]). It is possible to view the codes written by industry experts, thereby achieving capability to write standard codes (Jagadeesh et. al. [2008]). The whole community can share and modify this program and has proven to reduce the software development costs as well as the maintenance cost. Active open source projects usually have a well-defined community with common interests which is involved either in continuously evolving its related products or in using its results (Gacek and Arief [2004]).

Recently, software industry started adopting open source for development of these proprietary products. They use well developed open source libraries, tool chains or stripped down versions of already developed software modules for developing their own products. The open source model includes the concept of different concurrent agendas and differing approaches in production, in contrast with more centralized models of development such as those typically used in commercial software companies. Software systems are becoming increasingly complex as customers demand richer functionality delivered in even

shorter time scales (Pekka et. al [2009], Clark et. al. [2008]). Developing complex software systems using current code-centric technologies is difficult and expensive (France and Rumpe [2007]). As per the Standish Group, 84% software projects fail to deliver what has been promised on time and according to the budget (Greenfield et. al [2004]). Model driven development approach fastens the software development and is a solution for complex projects.

Some of the factors affecting reliability are the user groups involved, type and rate of failure, where the failure can be due to some planned or unplanned events or human activities, number of operational units of the software, time the user spends using the software, load on the underlying hardware, Free and Open Source Software (FOSS) testing, bug identification and testing, personnel expectation and analysis, community or individual members working on the distributed development of the FOSS (Kalpana et. al. [2011]) etc.

Research on the adoption of Information Technology(IT) innovations, has frequently drawn on innovation adoption theory (Bajaj [2000], Chau and Tam [1997], Cooper and Zmud [1986]). However, a weakness identified in many innovation adoption research has been an excessive focus on adoption at the individual level and not enough on the organizational level (Eveland and Tornatzky [1990], Eugene et. al [2005]).

The importance of FOSS is an important aspect of the study and hence a survey is planned to execute. Recently software industries

including Government organizations adopted FOSS for building their software based infrastructure for various reasons. There are very few studies done to evaluate the reliability of such community-driven software system products.

The main objective or the intention of the survey is :

- To find out how much is the use of open source software among different communities.
- Why community is moving towards FOSS and
- Why government and many of the IT firms are adopting FOSS.

The survey is conducted by preparing a questionnaire and is circulated to collect data, the questionnaire wass attached as APENDIX B. About 1000 records of data was collected. Which is processed by using simple JAVA codes and based on the processed data graphs were drawn and come to the conclusions.

4.2 How Free and Open Source Software (FOSS) helps the Project Manager

Project management involves planning, monitoring, and controlling of the people, processes and events that occur, as software evolves from a preliminary management concept to an operational implementation. (Pressman [2007]) They supervise the work to ensure that it is carried out to the required standards and also whether if the product is delivered to the customer in the stipulated time and within the budget. Good

management cannot guarantee project success. However, bad management usually results in project failure (Sommerville [2003]). The model helps the project manager to take appropriate decisions in time and thus it becomes a systematic approach. The project manager can make use of freely available open source models and codes for their development purpose. They can make use of or reuse certain codes that are available for developing their project modules. The logic available may be more comfortable and more simple for adopting to find out solutions for their project. So, automatically they can cut short their project cost as well as the time for development.

4.3 Integration of FOSS with Closed Source Software (CSS).

This is a new paradigm in software development and is a community based approach in the software development. Open source philosophy proved that it was able to produce software that was able to compete with commercially produced software [www.linux.org]. It is possible for the software engineers or developers to take advantage over the source code available over the net to find out their required logic to solve their problems, or to reuse the available code for their development purpose. Another thing is that the developers can make use of the help available on the net by the user groups formed over the net. The developers can post their doubts to get useful solutions. They will get different solutions out of which they can find out the most suitable ones. Even then the integration is slightly difficult. The main difficulty is in finding out the most suitable and reliable solutions.

If these situations can be tackled, this is the easiest and cost effective method to develop reliable software.

4.4 Pros and Cons of Free and Open Source Software Development.

Open source is assumed to be risky, even in some situations where it clearly provides more functionality (Mark [2004]). If we utilize the required code only as per our requirement, it may not be the case. Development becomes more easy. The interest in Open Source is increasing because of three principal issues like Risk, Cost and Functionality (Mark [2004]). Risk refers to the likelihood that the software will be stable and continue to meet needs over the long term. Cost means the overall cost of ownership, thus involving both licensing costs and support costs. Functionality pertains to the overall capability to meet specific operational requirements.

Some of the pros and cons of open source software are, reduced costs and less dependency on imported technology and skills, affordable software for individuals, enterprise and government, universal access through mass software rollout without costly licensing implications, access to government data without barrier of proprietary software and data formats, lowered barriers to entry for software business, participation in global network of software development, supplier independence, limiting vendor lock-in, and patches or updates become available quicker, which limits breakdown and security risks.

At the same time there are limitations and drawbacks to the use of open source software. They may include: Available support for open source software: In the past years support has been lacking a professional approach, Finding the appropriate software: Since FOSS is not 'advertised' it can be very difficult to select the appropriate applications for the task it has to support. A more active approach is needed from the users, Documentation: The documentation that accompanies FOSS software application is often idiosyncratic and some times non-existent. FOSS developers are motivated towards the technical aspects of the application than towards the usability. Limited best practices: There are very little known and documented cases of large scale migration from CSS to OSS and Hardware-software fit: FOSS often lags behind concerning new hardware. This is caused by the fact the hardware manufacturers fail to release hardware specifications in time to the FOSS community.(Victor and Corrado [2003]).

4.5 Case Study

A comprehensive survey has been conducted among the individual programmers, Government and many of the IT firms to identify the role of these communities towards the open source software and an analysis of the results are presented. For giving clarity to the study, the community has been categorized into three groups, common users, business industry and the government of each country. The view of these communities towards open source is discussed below.

4.5.1 Common User's Community

Common user's open source community consists of individuals or groups of individuals who contribute to a particular open source product or technology. The open source process refers to the approach for developing and maintaining open source products and technologies, including software, computers, devices, technical formats, and computer languages. For analyzing this community attitude towards FOSS, a survey has been designed to gather data on the factors influencing participant's satisfaction with free software. The factors considered include intended audience and their back ground who are the members of the community, participant's knowledge and level of people working on the open-source software, attitude towards open source software, user's experience on open-source software etc.

4.5.1.1 Intended Audience and their Back Ground - who are the Members of the Community

The users for the survey were mostly developers, employees and the college students who code or use the open source software for different purposes. The factors such as the level of education, age groups, gender and experience with the open source software were considered. And for that different levels of people at different levels were selected for the survey. The motive of the survey was to identify end users and programmers. From the survey concluded that open source software are mostly used in student community with the age below 25 as in Fig. 4.1.

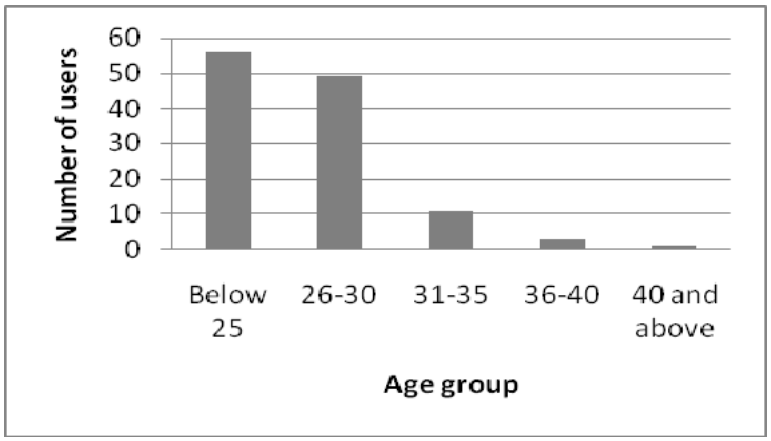


Figure 4.1 Participant's age group Vs No of users in the survey

4.5.1.2 Participant's Knowledge

To know how the level of knowledge influences the use of FOSS, was categorized the survey into different levels i.e. hardware, operating systems and programming languages. Knowledge levels were divided into three categories minimal, moderate and expert. The number of under graduate users were more compared to post graduate or high qualified users. About 10% were experts in hardware knowledge, 62% were experts in operating system, about 50% were having programming knowledge and only 7% were having high skill in design and development ability. The understanding of hardware, operating systems and programming languages by people having minimal, moderate and expert level of knowledge was measure/ surveyed. The survey revealed as in Fig. 4.2 shows that - open source software were not only for the expert users but also for novice users.

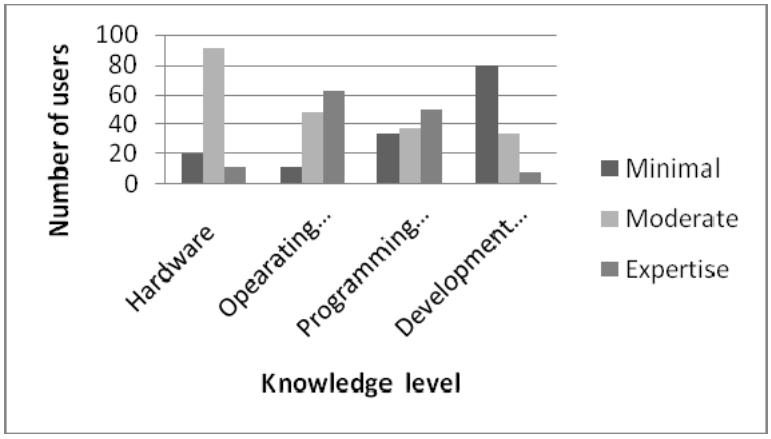


Figure 4.2 Knowledge level of users

The analyses of the above graph confirmed the following hypothesis that majority of the users do have moderate knowledge about computer hardware, but at the same time they have the expertise level of understanding of operating system.

4.5.1.3 Attitude Towards Open Source Software

This section mainly focuses the familiarization of open source with the end users. Users were quite familiar with the term "open source" before the survey. While choosing a new application most of the users give preference to open source software wherever possible, while some of them want to know if open source software will meet their needs or not. Most of the users, use open source software as much as possible on the computers provided by their employer. The main questions asked for the survey were, How an open source software makes difference than a closed source software? Why do users use open source software? What

operating systems do users use on the computers they own and the computers provided by the employers? How much preference do they give to open source software compared to closed source software?

In order to empirically investigate these questions, the investigator hypothesizes the following: While choosing a new application, most of the users give preference to open source software wherever possible, while some of them see if open source software meets their needs or not. For others it makes no sense whether the software is open source or proprietary. While some of them always choose open source software, most of the users, use open source software as much as possible on the computers provided by their employer. While others use sometimes. Besides some users prefer to use Open source software always on the computers provided by their employer.

To find out user attitude towards open source software the following questions were asked for the survey (Fig. 4.3). Are the users satisfied with the documentation provided, installation procedure- how easy is it, easy to add new features, is the community helpful , what about security and access control, are they free from bugs, what is its functionality and its reliability. Most of the users are satisfied with open source software, as it is free from bugs and has high security.

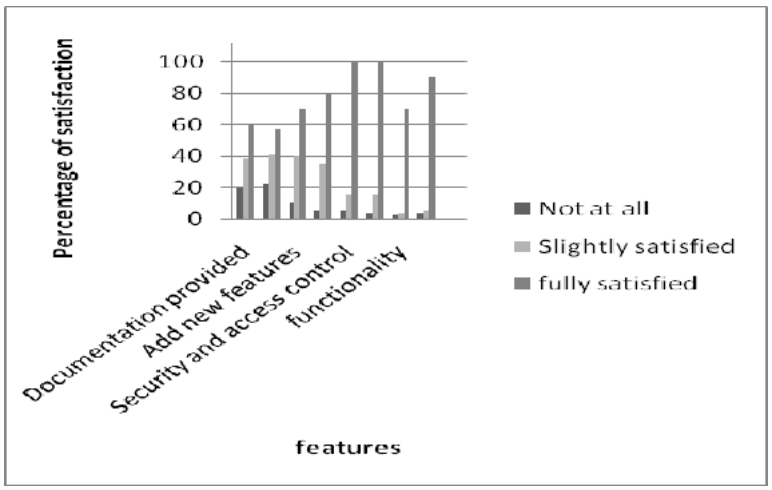


Figure 4.3 Users satisfaction towards the OSS features

4.5.2 Business Community

FOSS supports business oriented customization with the open-source features. Thereby, many firms today are integrating to these platforms to reduce their platform migration costs. From the survey it is observed how open-source software are helpful, what is their use and what sort of persons are using such technology. In business, mainly due to the following reasons, open-source software are more popular. One is Flexibility which is incensed in such a way that one can modify it according to the suitability and the specific needs of the business. The second is the reliability, since it will be having fewer bugs and it is more reliable. The third factor is the cost. It is free of cost, but sometimes one has to pay some money for the support. The fourth factor is longevity: When the commercial product's company is out from the business support cases then there will be no support. Instead, if open-source

software are used there are so many communities that are always in the forefront to help and support these products. The survey gives the following results as in Fig. 4.4.

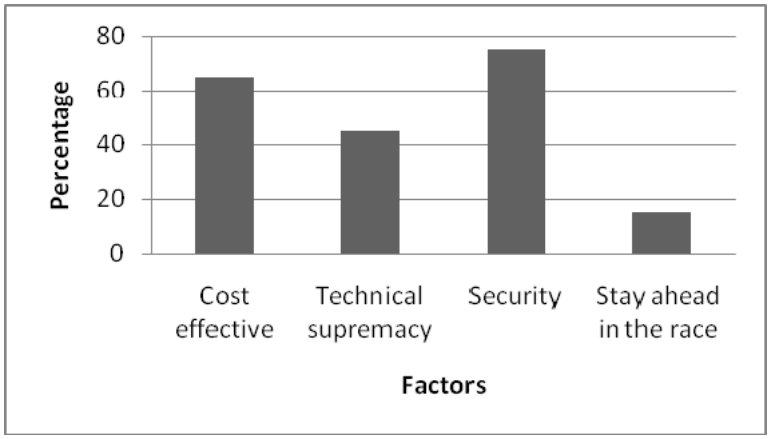


Figure.4.4 Factors Influencing the Business Industry

Community support for operating system like debian, Fedora are more in use about 65% than Paid support OS like RedHat (35%). This gives an idea regarding the importance and prominence of community support. The software that are not paid, but based on community support are universally accepted. The use of Open-source Software in the Industry is very high due to cost effectiveness (65%), Technical Supremacy (45%), Security & Networking (75%) and also they stay ahead in the race too (15%). This reveals that the cost is minimum while using open-source software. Security is also one of the prime reasons why people are using it in the business. This whole section tells how important is the open-source software either in terms of cost or

performance. Networking is the heart of business and with the help of these, open-source software are at the top in terms of satisfaction.

4.5.3 Government

This community includes different educational institutions and the, administrative departments of government of each country. Government is one of the major ICT (Information and communications technology) and software consumers worldwide. Government agencies wishing to develop an understanding of the open-source software's potential, might look no farther for information than a non-profit trade association. As in the private sector, the types of open source software the government is pursuing are becoming more sophisticated [Eugene Glynn, Brian Fitzgerald et al 2005]. Now the Government of Malaysia proudly reports an astonishing 97% adoption rate for open source software. The Open Source Observatory and Repository for European public administrations (OSOR.eu) supports and encourages the collaborative development and re-use of publicly-financed free, libre and open source software (FOSS) applications developments for use in European public administrations. Governments too, have begun to take notice of this phenomenon. Countries such as Brazil, China, Malaysia, South Africa, and Viet Nam, are implementing nationwide policies or legislation promoting FOSS. While the often-cited cost and stability benefits of FOSS are attractive, governments often choose to promote FOSS in their own countries for a variety of other reasons.

Countries like Singapore have offered tax reductions to companies that use the GNU/Linux operating system. The guaranteed cost-saving makes FOSS systems more attractive. A consolidated survey of total FOSS usage of the countries India, UK, US and Malaysia has been done (Dominik Richter, Hangjung Zo, et. al. [2009], http://www.govtalk.gov.uk/policydocs, www.egovos.org, Kenneth Wong [2004]). The information is not very accurate since it was received from the websites of the countries and other related reports. Fig.4.5 shows the overall report. The policy of each country is discussed.

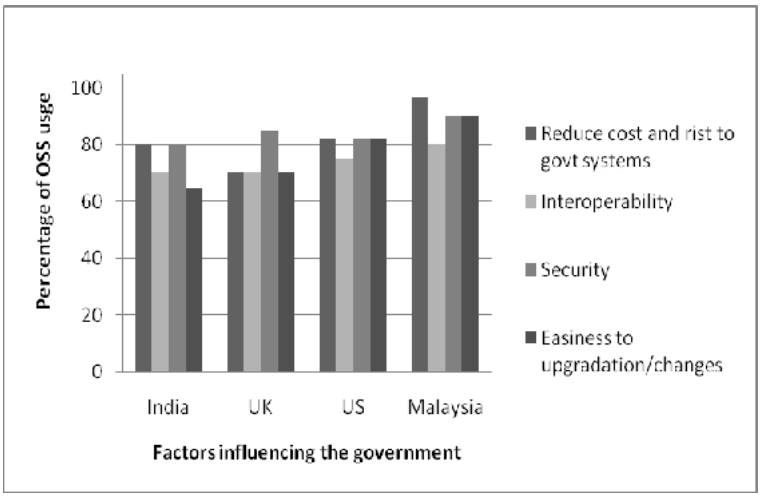


Figure 4.5 Factors influencing different Government to adopt FOSS

The department of Information Technology is developing, supporting and promoting Open Source Software in India. Advantages like increasing interoperability, reducing costs, achieving vendor independence, enabling localization, reducing piracy/copyright infringements and

increasing growth of knowledge-based society are among the compelling reasons for adopting FOSS in India. In many cases it has been observed that FOSS is functionally and qualitatively equivalent to or even superior than the proprietary products. The initiatives toward offering a low cost computing, flexibility and choice to the end users include Bharat Operating System Solutions (BOSS) desktop and server versions and EduBOSS for schools. BOSS is a GNU/Linux based localized Operating System distribution that supports 18 Indian languages. FOSS elective courses are part of curriculum in several higher institutes of technical learning.

The policies of the UK government to adopt FOSS are: the government will consider FOSS solutions alongside proprietary ones in IT procurements and contracts will be awarded on a value for money basis. The Government will only use products for interoperability, seek to avoid lock-in to proprietary IT products and services. Publicly funded R&D projects which aim to produce software outputs shall specify a proposed software exploitation route at the start of the project. At the completion of the project, the software shall be exploited either commercially or within an academic community or as FOSS.

The U.S. Government has the ability to promote and continue the development of open source software through its purchasing policies. Their policies include added reliability and security which FOSS products provide, and the increased demand for these products would encourage more corporations and independent programmers to embrace

FOSS methods. The U.S. Postal Service, for example, uses a highly modified version of Linux to read addresses on envelopes electronically. Some other agencies use Linux for network administration tasks, as it is considerably more affordable then the competing Windows Software. Another action involves the vast pool of software created for internal tasks within the government and the military. Collecting non classified source code in a series of repositories for the purpose of allowing public access would benefit both government and the public. Companies and individuals will have access to the expertise of government and military software engineers, obviating the need to solve software problems which have already been solved. Additionally, if some individual or group takes an interest in improving some piece of software in use in a government agency, the agency will reap the benefit, at no cost to taxpayers. An attacker can find security flaws in software with or without the source code. Opening source code to the public, though it may create short-term apprehensions, will result in more secure software in the long run.

The Malaysian government is using 97% open source software [http://www.mit.gov.in]. The report reveals that 703 of 724 agencies in Malaysia have switched to FOSS. The Malaysian government is using OpenOffice, MySQL, Apache web server and a Linux based distro. The government produces regular guidelines and provides advisory services to assist public sector agencies, when implementing FOSS. Policies in general are discussed. FOSS Adoption should be based on the least

disruptive and fit for purpose implementation. FOSS procurement should be based on merits, value for money, transparency, security and interoperability, as well as in accordance with the Government procurement policies and procedures. FOSS education should be introduced through structured programs in school IT labs for primary, secondary and tertiary education levels.

4.6 Analysis

Free and open source software is gaining rapid growth after the increased popularity and use of internet. This has formed a different community of software developers all across the world. A measure of success and failure of an OSS can be considered as activities on the online community. Both government and business organizations have some concerns about the quality of the software. In organizations having problems with software defects, threats etc. Software defect management is not very easy without the community support. We observed that technically rich online forums are the corner stone of managing software defects in OSS. The management of new defect right from the identification, solution to fixing phases is communicated via online forum, which may give rise to problems such as improper communication, and others. A thorough analysis is needed in the design phase itself to reduce the problems with OSS after development. Here we propose a more efficient defect free model.

4.7 Importance of Open Source Software.

The usage of open source software is much more advanced and people are using it for the development of their own products. According to Zhou and Davis [2005], the Information Technology (IT) community is getting more used to applying open source solutions and in recent surveys, the majority of companies are found to be using open source software commonly as server operating system, web server and for web development. Apache web server software and Linux operating system are the most famous open source products, which have proven their quality and reliability. Even though these two products proved the success of open source products, people are still confused about using open source software products. This hesitation is present in private sector, government and business people as well. Ray[2004], former oracle executive, stated that the lack of formal support and velocity of change are the two common apprehensions. Hence, we can conclude that these hesitations are all originated from the quality, reliability and security levels of open source software and their evaluation. Further before concluding the major findings of this case study are

- FOSS are mostly used in student community.
- FOSS are not only for the expert users but also for novice users.
- Most of the users are satisfied with FOSS, as it is free from bugs and having high security.

- There are so many communities that are always in forefront to help and support FOSS products.
- The guaranteed cost saving makes FOSS system more attractive.

4.8 Conclusions

A case study on the use of open source software among different communities is discussed. Companies and governments have chosen Free and Open Source Software due to lower costs, independence from software manufactures, ability to modify etc. The survey shows that the community is moving towards FOSS due to its simplicity and support. Companies should re-evaluate FOSS when considering upgrades of their software due to the fast paced IT landscape. Since the process model followed by open source software is different from the closed source models a thorough analysis is required by the project managers before adopting the open source. Government and many IT firms are adopting open source software for building their custom products, since it offers independence and opportunities of innovation.

There are only a few studies on reliability aspects are carried out compared to closed source software development. To assure the quality of the developed product using open source scenario it is important to consider reliability aspects of the development process. Hence our problem was to conduct a study on reliability of open source software. It is also noted that only limited studies are carried out to find out the

reliability aspects of combined hardware and software components. So we incorporated both of these factors and decided to develop a model that incorporates both hardware and software components.

Chapter 5

Reliability of Open Source Software Projects

Contents

5.1 Introduction

Evaluation of quality of open source projects is important and there are no major method or metrics that are available. Currently, people make judgments on open source products based on relatively arbitrary criteria. There is a close relationship between the quality of a software product and stabilization of the bugs raised from the software product. Here, an attempt has been made to predict latent bugs in open source projects using time series analysis. We collected information regarding newly opened bugs of popular open source Packages. The trend of bug arrival rate has been collected from the concerned web sites of these projects and historical time series plot has been used to predict future values. The results are compared with open source reliability growth

models and a conclusion is made on the possible reliability growth models for open source projects.

Open source is emerging as a potentially important competitive force in the software industry, capturing the attention of venture capitalists and computing industry executives. Open source development is an area where people develop and distribute their products by downloading free source code available under a license. Open source software development has created an interest in the development circle. They make use of this environment to create quality products. The open source development environment is a different approach, where the main functionality is developed by the initiator and then made available for others to test, use and modify. Mistakes in the software are not considered problematic, but are accepted. Since the source code is distributed, every software engineer can change or extend the original product. So, where proprietary software is developed in- house and then released, open source software is under constant development because anyone in the world can change the code (Victor and Corrado [2003]). Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment (Michael [2007], Musa and Iannino [1981]). The reliability at any instant in a software system depends on the faults that exist in the software and the exposure of such faults through activation of the software via testing, usage etc.

5.2 Background

Usually there are two approaches for predicting software reliability. They are White-box and Black –box models. White-box models measure the quality of the system based on the structure and design of the product, where as in Black-box models the entire software system is treated as a single entity, thus ignoring software structures and component interdependencies (Cobra et. al[2009], Cheung [1980], Gokhale et. al [1998], Wang et. al [1999], Yacoub et. al [2005]). These models measure and predict software quality in the later phases of software development. The model rely on the data collected over an observed time period. Some examples of this type includes Yamada, S-Shape, Littlewood-Verrall, Jelenski-Moranda, Musa-Okumotto and Goel-Okumoto (Goel and Okumoto [1979], Littlewood and Verrall [1974], Littlewood and Verrall [1973], Musa and Okumoto[1984], Yamada et. al [1984]). Black-box reliability approach has been concentrated upon, to measure and compare the reliability of the selected OSS projects.

A fault or bug is a defect in the software that has the potential to cause the software to fail. An error is a measured value or condition that deviates from the correct state of software during operation. A failure is the inability of the software product to deliver one of its services. Therefore, a fault is the cause for an error, and software that has a bug may not encounter an error that leads to a failure. Failure behavior can be reflected in various ways such as Probability Density Function (PDF)

and Cumulative Distribution Function (CDF). PDF, denoted as $f(t)$, shows the relative concentration of data samples at different points of measurement scale, such that the area under the graph is unity. CDF, denoted as $F(t)$, is another way to present the pattern of observed data under study. CDF describes the probability distribution of the random variable, T, i.e. the probability that the random variable T assumes a value less than or equal to the specified value t. In other words,

$$F(t) = P(T \le t) = \int_{-\infty}^{t} f(x)d(x) \Rightarrow f(t) = F'(t)$$

Therefore, $f(t)$ is the rate of change of $F(t)$. If the random variable T denotes the failure time, $F(t)$, or unreliability, is the probability that the system will fail by time t. Consequently, the reliability $R(t)$ is the probability that the system will not fail by time t (Pham [2000]), i.e.

$$F(t) = P(T > t) = \int_{t}^{t} f(x)d(x) \Rightarrow R(t) = 1 - F'(t)$$

The reliability function of Weibull distribution is (Neubeck [2004])

$$R(t) = e - \left(\frac{t}{\infty}\right)^{\beta} \text{ ---------- (5.1)}$$

Weibull distribution is the most widely used distribution model. The 2-parameter Weibull distribution is widely used due to its ability to describe failure modes like initial,random and wear-out. Two special

forms of Weibull distribution are Rayleigh and exponential distribution (Kan[2003], Lawless[2003]).

The 2-parameter Weibull distribution has a probability distribution function of the form

$$f(t)=\lambda\beta(\lambda t)^{(\beta-1)}\exp\left(-(\lambda t)^{\beta}\right) \text{---------- (5.2)}$$

Where t represents time; $\alpha = 1/\lambda$ represents the scale parameter of the distribution and β represents the shape parameter of the distribution. The Weibull probability density function is monotone decreasing if $\beta <= 1$ and becomes bell shaped when $\beta > 1$. The larger the β value the steeper the bell shape. Its special case Rayleigh distribution has $\beta = 2$, while exponential distribution has $\beta = 1$. Fig. 5.1 shows the Weibull PDF for several values of the shape parameter (Kan[2003]).

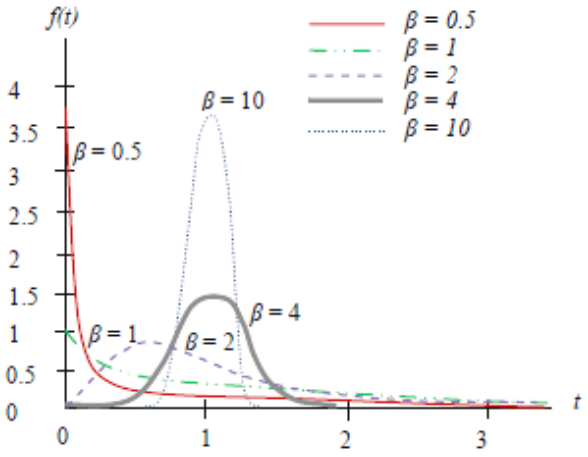


Figure 5.1 Weibull PDF for several shape values when α =1

In software quality engineering, large body of empirical data supports the finding that software projects follow a life cycle pattern described by Rayleigh curve. This is considered as a desirable pattern since the bug arrival rate stabilizes at a very low level (Zhou and Davis[2005]). In closed source software, the stabilizing behavior is usually an indicator of ending test effort and releasing the software to the field. The bug arrivals usually peak at the code inspection phase and get rather stabilized in the system test phase (Kan[2003], Zhou and Davis[2005]).

5.3 Data Collection and Analysis

Data were collected from sources like Debian.org, Bugzilla.org and Apache.org. The collected data processed with simple JAVA Codes. In the bug-analysis step, the frequency of bugs in a two week periods is calculated. Therefore, the x-axis and y-axis represent the biweekly time and the corresponding bug frequency, respectively. The bug arrival frequency for six projects are plotted and shown in the following Fig. 5.2.

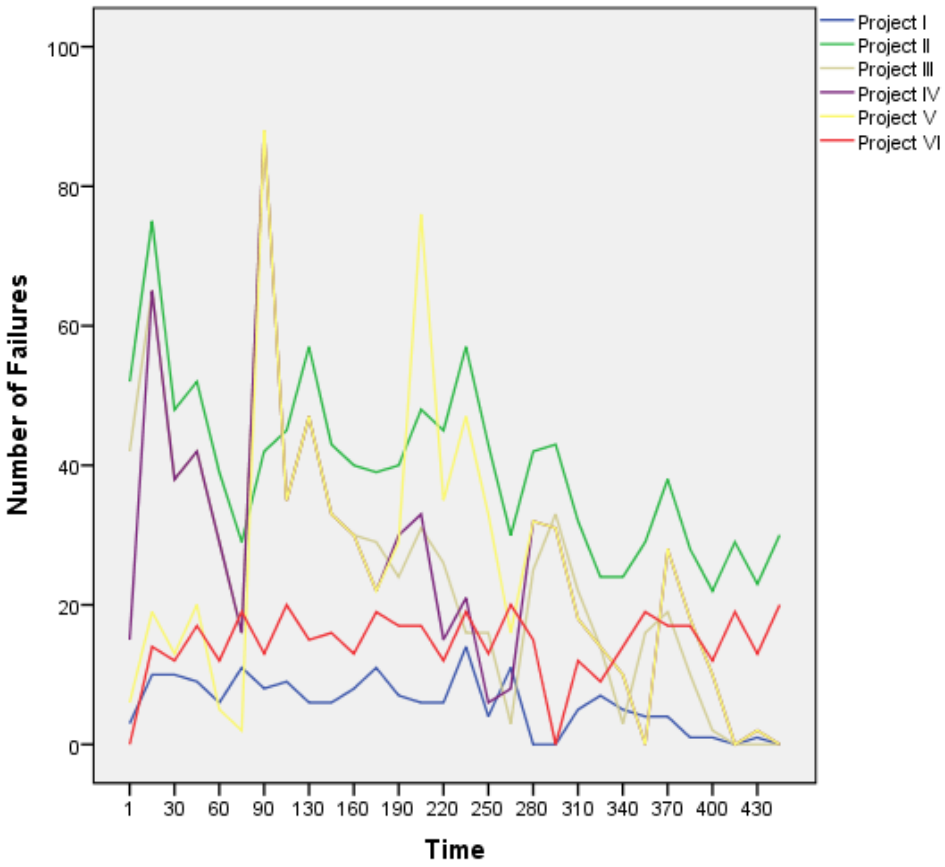


Figure. 5.2 Bug Arrival Frequency for Six Projects

In all the above projects it is seen that the frequency of bug arrival is slowly increasing and comes to a peak, then slowly decreasing and further it comes to a stable state. Nonlinear regression procedure is employed to model the data and to estimate the parameters.

Table 5.1 Estimated Parameters and R-square

Project	Lambda	Beta	R-square
Project I	0.0008	0.585	0.785
Project II	0.001	1.550	0.228
Project III	0.002	0.695	0.287
Project IV	0.035	1.311	0.348
Project V	0.029	2.292	0.363
Project VI	0.027	1.699	0.279

The Table 5.1 lists the estimations of the shape and scale parameters of the projects identified. Actual names of the projects were not revealed to follow standard software engineering ethics (Emam [2001]). The R-square value indicates good fit for some projects such as project 1, project V, project IV and project III. The failure rate and the corresponding predicted failure rate are indicated in the following Fig. 5.3 to 5.8. The Weibull distribution is fitted to arrive at the predicted failure rate.

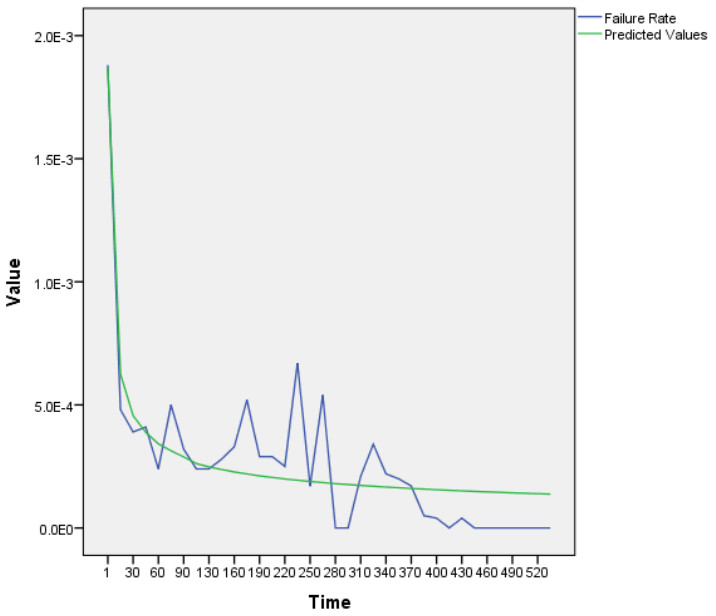


Figure 5.3 Failure Rate and Predicted – Project I

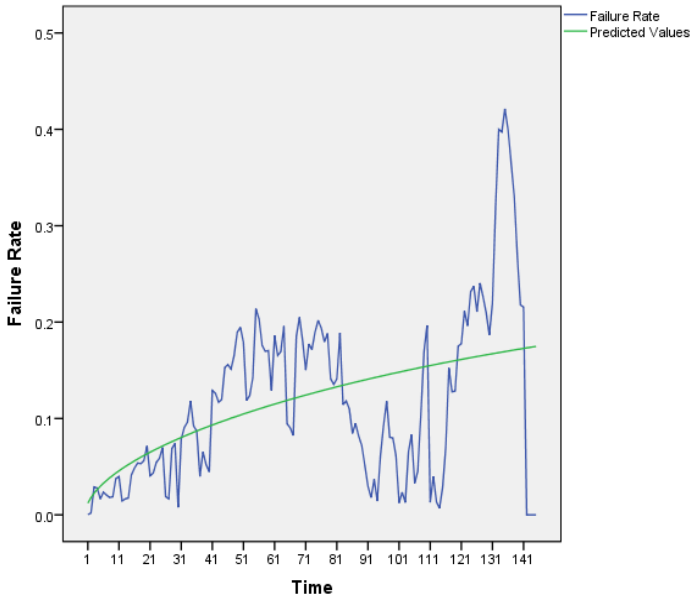


Figure. 5.4 Failure Rate and Predicted – Project II

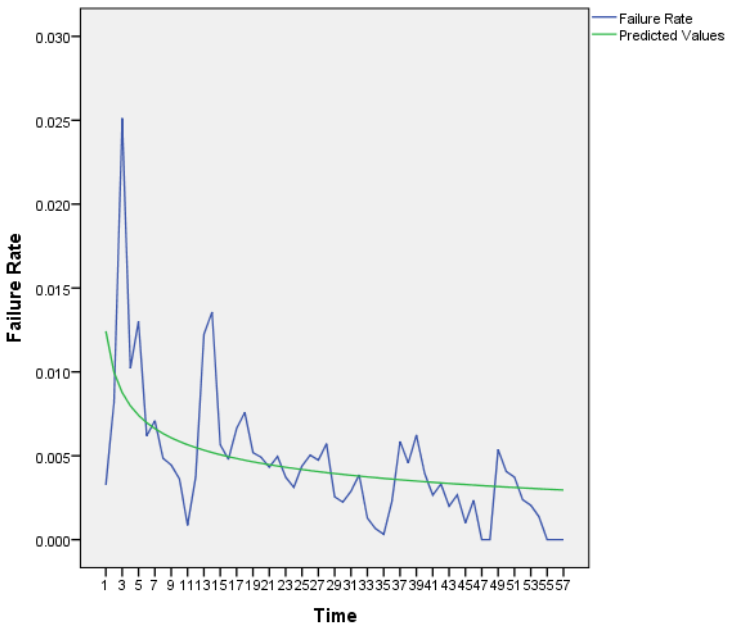


Figure 5.5 Failure Rate and Predicted – Project III

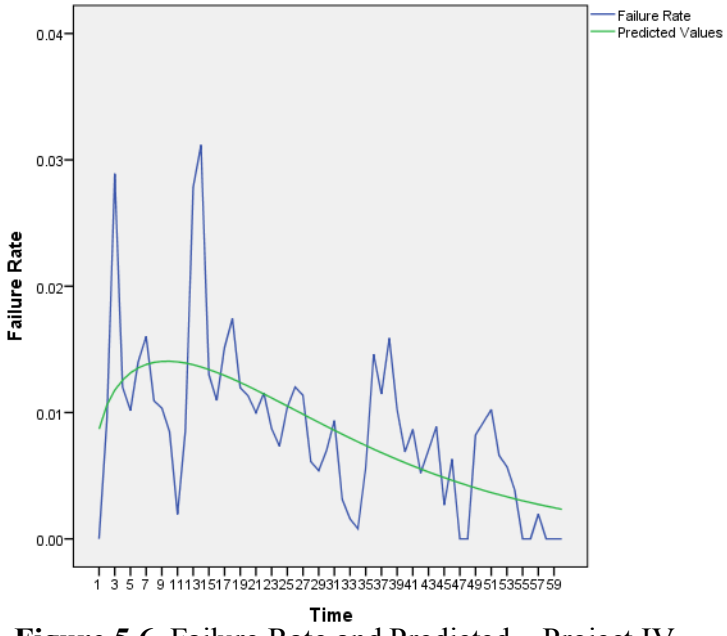


Figure 5.6 Failure Rate and Predicted – Project IV

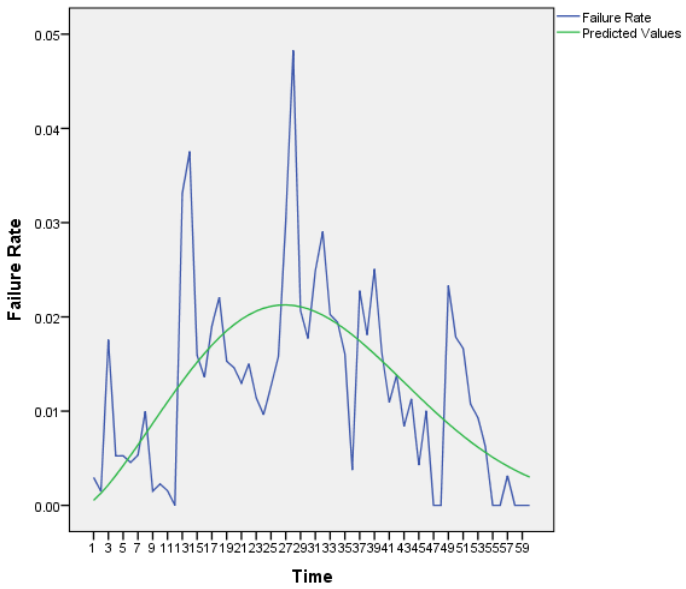


Figure 5.7 Failure Rate and Predicted – Project V

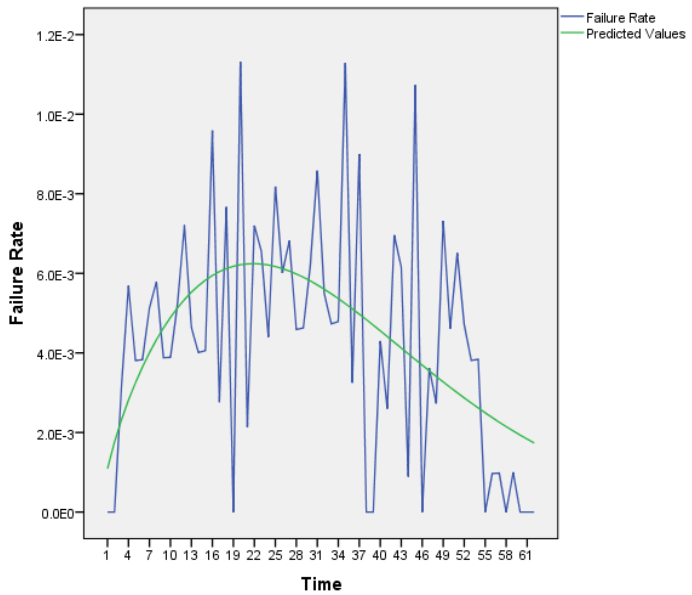


Figure 5.8 Failure Rate and Predicted – Project VI

It is observed that the bug frequencies for these projects appear to follow a pattern that can be represented by the Weibull distribution function with different β values. The projects I and III shows β values lesser than 1. This lower coefficient value indicates that the projects are in the initial stage of development and more and more bugs are reported with in a short span of time. Where as projects II, IV,V & VI show β values higher than 1 and closer to 2. This is considered as a desirable pattern since the bug arrival rate decreases exponentially and stabilizes as time progress. From the literature we found that this pattern is supported by large body of empirical studies and in software projects that follow a life cycle pattern similar to Weibull distribution. This pattern is also supported by Musa-Okumoto model in which possible bugs are captured in the early stages of development. The rate of undetected bugs are significantly less and is decreased exponentially. An implication of this behaviour is that as the project evolves in time there are less frequency of reports on bugs and probably it is difficult to detect and fix.

Further reliability of the projects can be calculated by inserting the shape and scale parameters from Table 5.1 into the Weibull reliability function in (5.1). Fig. 5.9 exhibits the reliability graphs for the five OSS products. As shown Project II and Project III has the highest reliability and the other projects show almost same decreasing reliabilities.

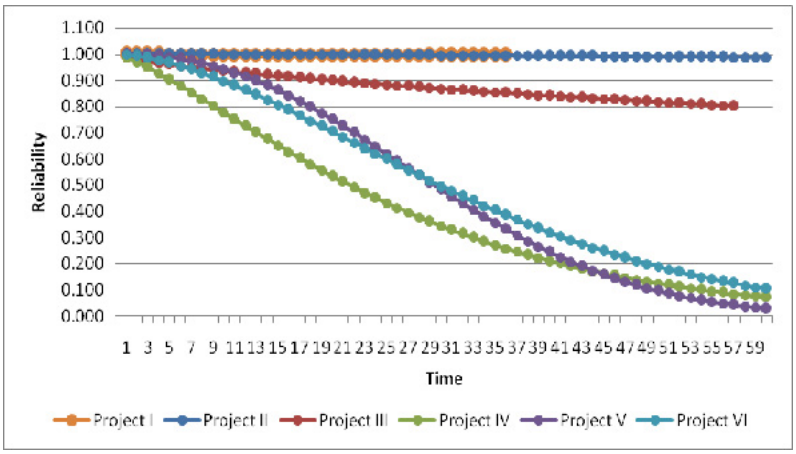


Figure. 5.9 Time VS Reliability for different projects

A comparison of reliabilities for different projects is carried out by calculating reliabilities using theory, Musa model and Weibull distribution for these six products. These are plotted as in Fig. 5.10 to Fig. 5.15. It is obvious that the Weibull model is appropriate for modeling open source products.

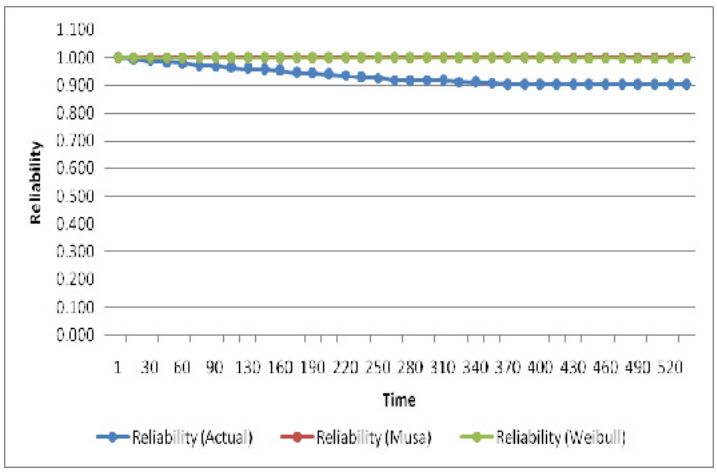


Figure 5.10 Time VS Reliability Project I

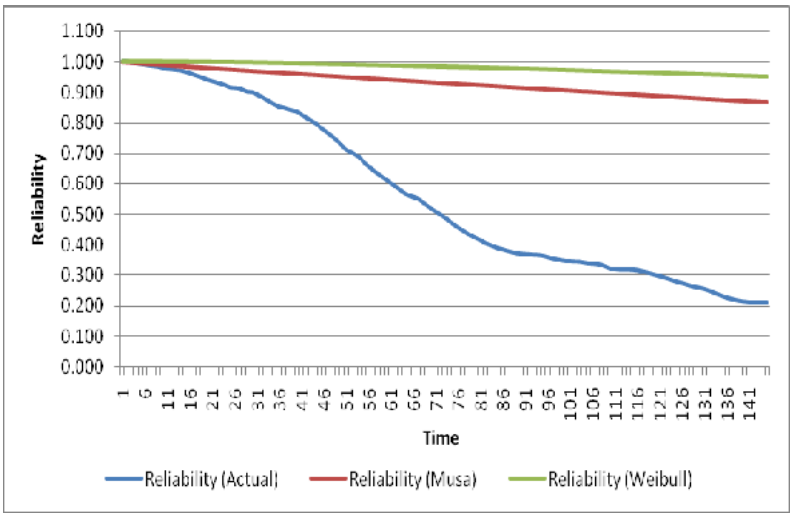


Figure 5.11 Time VS Reliability Project II

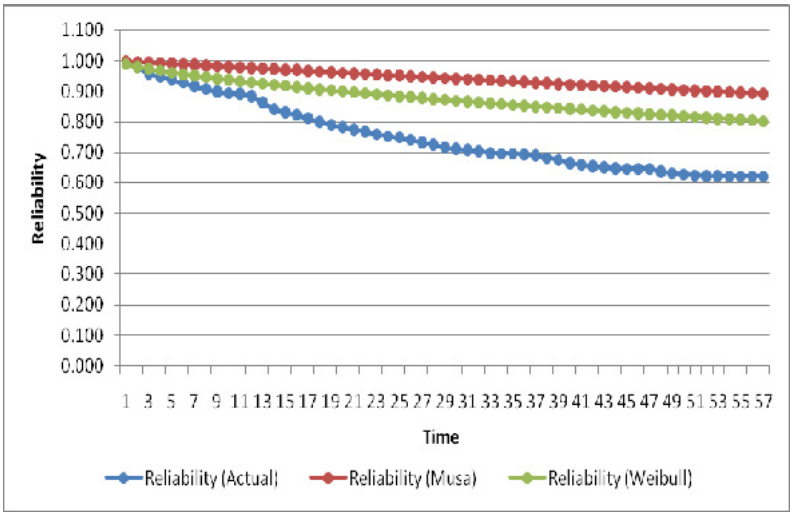


Figure 5.12 Time VS Reliability Project III

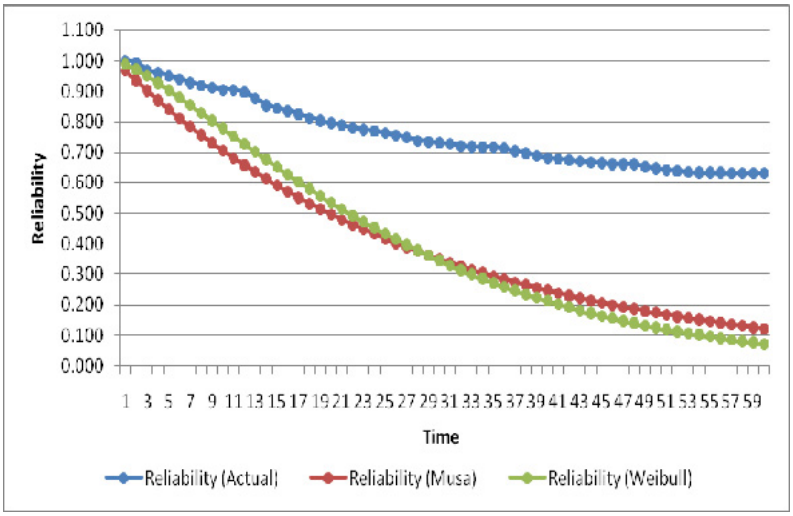


Figure 5.13 Time VS Reliability Project IV

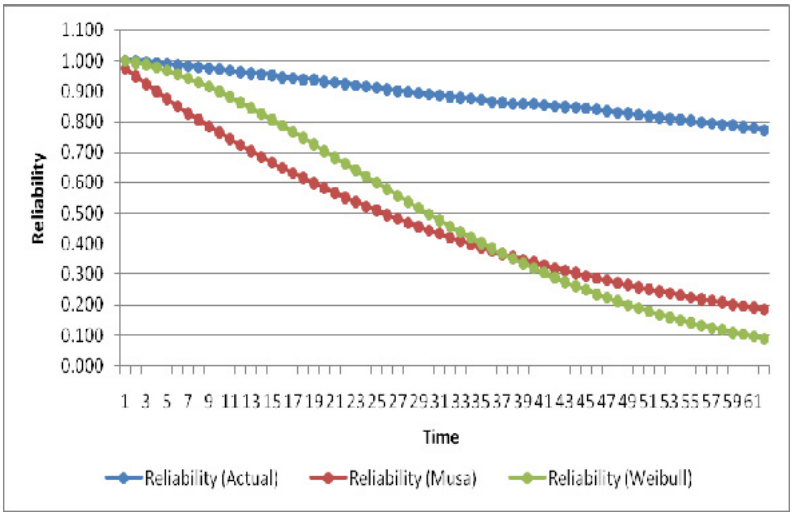


Figure 5.14 Time VS Reliability Project V

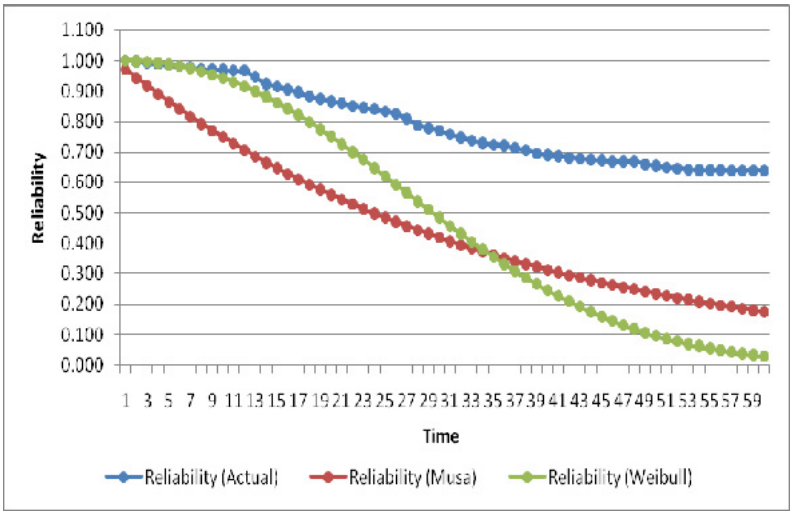


Figure 5.15 Time VS Reliability Project VI

It is evident from the above figures that General Weibull distribution is a possible way to define a reliability model. Estimations of shape parameters from various open source projects are different indicating that in contrast to closed source projects it is unlikely to find a special case to model all open source projects. It might be a better way to model individual open source projects separately, further time series analysis would be appropriate for predicting latent bugs in individual open source projects.

5.4 Conclusion

Bug tracking data was collected from a few popular open source projects and the time related bug arrival was investigated. Bug arrivals of most OSS projects will stabilize at a very low level and the stabilizing

point can be viewed as the mature point for adoption consideration. The general Weibull distribution offers a possible way to establish the reliability model.

…..ஜ۞ஜ…..

Chapter 6

A Simplified Model for Evaluating Reliability of a Computing System

Contents

6.1 Introduction

Evaluation of Reliability of a computing system is important for predicting possible system failures in the future. The probability of a device giving satisfactory performance for a specified period under specified operating conditions is the reliability. When a unit or system does not perform satisfactorily, it is said to have failed. The first step in reliability analysis is to understand the pattern of failure and this can be obtained from life test results. That is, by testing a fairly large number of models until failure occurs, and observing the failure rate characteristics as a function of time. It is necessary for the analysis to link reliability with experimental or field – failure data. These data will also provide a

basis for formulating or constructing mathematically, a failure model for general analysis. Further, a model is developed to estimate the reliability of computing systems by incorporating software and hardware elements. The main disadvantage with the existing models is that computational system reliability analysis is purely focussed on software alone. No attempt has been made to incorporate hardware reliability in computational system reliability analysis. The present work brings out a simplified model for computational system reliability evaluation by incorporating hardware as well as software failures. A separate algorithm is also developed for software reliability estimation.

6.2 Reliability Measures

The reliability definitions given in the literature vary among different practitioners as well as researchers.

Mathematically, reliability $R(t)$ *is* the probability that a system will be successful in the interval from time 0 to time t:

$$R(t) = P(T > t), \quad t \geq 0 \text{ ---------- (6.1)}$$

where T is a random variable denoting the time-to-failure or failure time.

Unreliability $F(t)$, a measure of failure, is defined as the probability that the system will fail by time t:

$$F(t) = 1 - R(t) = P(T \leq t) \; for \; t \geq 0$$

In other words, $F(t)$ *is* the failure distribution function. If the time-to-failure random variable T has a density function *f(t),* then

$$R(t)=\int_0^{\infty} f(t)dt$$

or, equivalently,

$$f(t)=-\frac{d}{dt}[R(t)]$$

The density function can be mathematically described in terms of T:

$$\lim_{\Delta t\to 0} P(t<T\le t+\Delta t)$$

This can be interpreted as the probability that the failure time T will occur between the operating time t and the next interval of operation, $t+\Delta t$.

If the time to failure is described by an exponential failure time density function, then

$$f(t)=\frac{1}{\phi}e^{-t/\phi}\quad t\ge 0,\phi>0$$

and this will lead to the reliability function

$$R(t)=\int_t^{\alpha}\frac{1}{\phi}e^{-\frac{t}{\phi}dt}=e^{-t/\phi}\quad t\ge 0$$

Thus, given a particular failure time density function, or failure time distribution function, the reliability function can be obtained directly.

A system or a complex product is an assembly of a number of parts or components. The components may be connected in series or in parallel, or it may be a mixed system, where the components are connected in series as well as in parallel. Reliability Block Diagrams (RBD) were used to measure the system reliability by assigning failure rates to each of the constituent components comprising the system (Bream and Curator [1995]). Here, the same approach is extended to the system which includes hardware and software components together.

If the components of an assembly are connected in series, the failure of any component causes the failure of the assembly or system.

The following Fig. 6.1 shows a system consisting of *n* units which are connected in series.

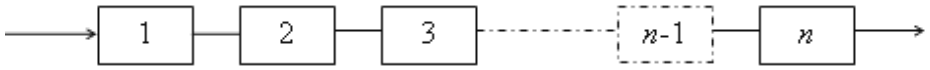


Figure 6.1 Reliability block diagram of a system having n components connected in series(Xie et al.[2004]).

Let the successful operation of these individual units be represented by $X_1, X_2, \cdots\cdots\cdots\cdots, X_n$ and their respective probabilities by $P(X_1), P(X_2), \cdots\cdots\cdots, P(X_n)$. For the successful operation of the system, it is necessary that all *n* units function satisfactorily. Hence, the

probability of the simultaneous successful operation of all units is $P(X_1 and X_2 and \cdots\cdots\cdots\cdots and . X_n)$. Therefore according to the multiplication rule,

$$R(t) = P(X_1 and X_2 and \cdots\cdots\cdots\cdots and . X_n)$$

$$R(t) = P(X_1) \times P(X_2/X_1) \times P(X_3/X_1 and X_2) \times \cdots\cdots\cdots\cdots \times P(X_n/X_1 and X_2 \cdots\cdots\cdots and X_{n-1})$$

In this expression, $P(X_2/X_1)$ represents the probability of the successful operation of unit 2 under the condition that unit 1 operates successfully. Similarly, $P(X_n / X_1 and X_2 and \cdots\cdots\cdots X_{n-1})$ represents the probability of the successful operation of unit *n* under the condition that all the remaining units 1,2,3,……,*n*-1 are working successfully. If the successful operation of each unit is independent of the successful operation of the remaining units, then events $X_1, X_2, \cdots\cdots\cdots\cdots, X_n$ are independent and the above equation becomes

$$R(t) = P(X_1) P(X_2) \cdots\cdots\cdots\cdots\cdots\cdots P(X_n)$$

That is, $R(t) = R_1 R_2 R_3 \cdots\cdots\cdots\cdots R_n$,

where $R_1 R_2 R_3 \cdots\cdots\cdots\cdots R_n$ are component reliabilities.

Several systems exist in which successful operation depends on the satisfactory functioning of any one of their *n* sub-systems or elements. They are said to be connected in parallel. The following Fig. 6.2 shows a system consisting of *n* units which are connected in parallel.

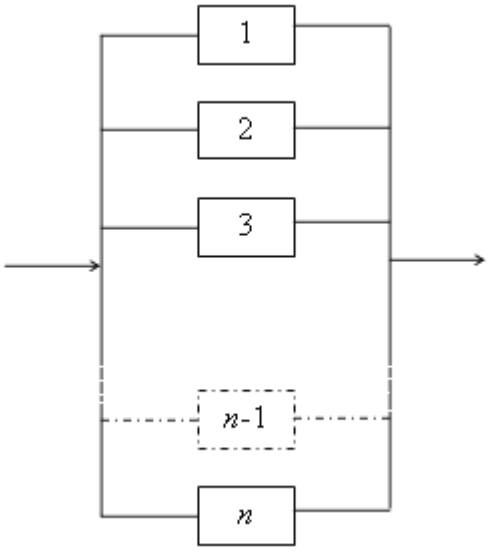


Figure 6.2 Reliability block diagram of a system having n components connected in parallel(Xie et al.[2004]).

Let $X_1, X_2, \cdots\cdots\cdots\cdots, X_n$ represent the successful operation of units 1,2,……,*n* respectively. Similarly, let $\overline{X}_1, \overline{X}_2, \cdots\cdots\cdots\cdots, \overline{X}_n$ represent the unsuccessful operation. If $P(X_1)$ is the probability of successful operation of unit 1, then $P(\overline{X}_1)$ is the probability of its failure. Further, $P(\overline{X}_1) = 1 - P(X_1)$

For the complete failure of the system, all *n* units have to fail simultaneously. If $F(t)$ is the probability of failure of the system, then

$$F(t) = P(\overline{X}_1 and\, \overline{X}_2 and \cdots\cdots\cdots\cdots and . \overline{X}_n)$$

$$F(t) = P(\overline{X}_1) \times P(\overline{X}_2 / \overline{X}_1) \times P(\overline{X}_3 / \overline{X}_1 and \overline{X}_2) \times \cdots\cdots\cdots\cdots \times P(\overline{X}_n / \overline{X}_1 and \overline{X}_2 \cdots\cdots\cdots\cdots and \overline{X}_{n-1})$$

In this expression, $P(\overline{X}_3 / \overline{X}_1 and \overline{X}_2)$ represents the probability of failure of unit 3 under the condition that units 1 and 2 have failed. The other terms can also be interpreted in the same manner. If the unit failures are independent of each other, then

$$F(t) = P(\overline{X}_1)P(\overline{X}_2)\cdots\cdots\cdots\cdots P(\overline{X}_n) = 1 - R(t)$$

$$\therefore R(t) = 1 - \{(1 - P(X_1))((1 - P(X_2))\cdots\cdots\cdots(1 - P(X_n))\}$$

That is, $R(t) = 1 - (1 - R_1)(1 - R_2)\cdots\cdots\cdots(1 - R_n)$

where $R_1 R_2 R_3 \cdots\cdots\cdots\cdots R_n$ are component reliabilities

If a system is having a mixed configuration, then it will have components connected in parallel as well as in series.

Typical approaches to achieve higher system reliability are:

(1) increasing the reliability of system components and

(2) using redundant components in various subsystems in the system (Kuo and Prasad [2000] , Hsieh et. al. [1998]).

In the reliability literature, these methods are commonly posed as reliability optimization problems. Depending on the choice of decision variables, creating redundancy (adding parallel units), increasing component's reliability or both, the reliability optimization problem can be formulated as a redundancy allocation, a reliability allocation or a mixed optimal problem, respectively. Information on different formulations and solution procedures are presented by Kuo et. al. [2001].

6.3 Probability Density Function

Failure density is the ratio of the number of failures during a given *unit interval of time* to the total number of items at the very beginning of the test (also called as initial population).

Reliability is the ratio of survivors at any given time to the total initial population. As time tends to infinity reliability tends to zero and in terms of failure density function this can be expressed

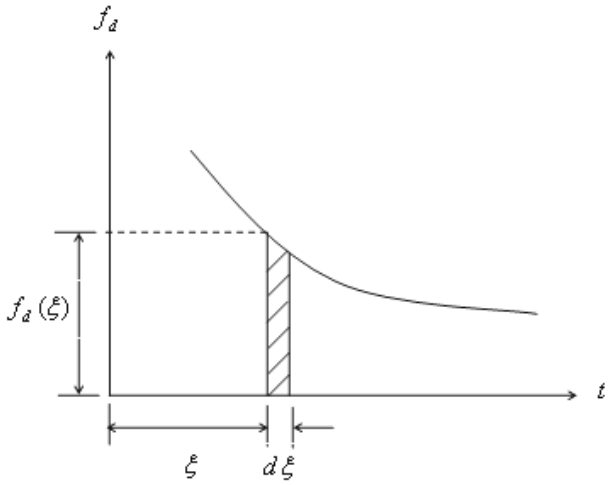


Figure 6.3 Probability Density Function

with reference to the above Fig. 6.3 as , $\int_0^T f_d(\xi)d\xi = 1$ ----------(6.2)

(where ξ is a dummy variable)

That is, the probability that a specimen will fail at time t = $\infty \cdot$ is 1 (that is, a certainty).

In probability theory the function $f_d(\xi)\, d\xi$ is known as probability density function.

Reliability in terms of failure density and failure rate can be expressed as

$$R(t) = 1 - \left(f_{d_1} + f_{d_2} + \cdots\cdots\cdots + f_{d_t}\right) = 1 - \int_0^t f_d(\xi) d\xi \quad \text{-----(6.3)}$$

Failure rate can be defined as the ratio of number of failures during a particular *unit time interval* to the average population during that interval. Failure density in terms of failure rate and reliability can be expressed as

$$f_d(t) = Z(t)R(t) \quad \text{----------(6.4)}$$

Reliability of an individual component in terms of failure rate can be expressed as

$$R(t) = e^{-\int_0^t Z(t)dt} \quad \text{----------(6.5)}$$

For a component with a constant failure rate equation (6.5) reduces to

$$R(t) = e^{-\lambda t} \quad \text{----------(6.6)}$$

The constant failure rate model is widely used in the literature to reduce the computational burden of the resulting problem [Goel et al, 2002] because the parameter MTBF [Shouri , Sreejith 2008], which is

the average time between failures, obtained from equation 6.7 becomes time-independent in this case.

$$MTBF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda} \quad \text{----------(6.7)}$$

6.4 An Algorithm for Estimating Software Reliability

An attempt is made to develop a systematic procedure for evaluating the reliability of open source software based computing system by considering the prevailing trends in industry. The methodology involves defining an equation for the pattern of failure based on the available bug arrival rate and developing a generalized model for the reliability of the software. A flowchart for the procedure is as shown in Fig. 6.4.

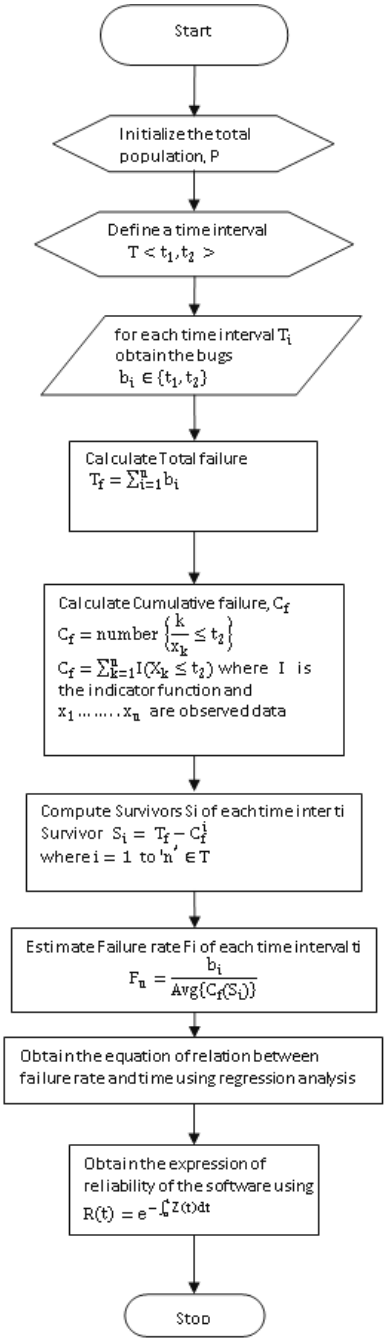


Figure 6.4 Flowchart for the systematic procedure

The following are the assumptions involved in the analysis (Isitan et. al.[2011], Crowston and Scozzi [2002]).

1) The software analyzed is open source.

2) As the open source software is made up of a very large community the environmental changes are not considered.

3) The total number of packages at the beginning of the analysis is assumed to remain constant and is taken as the initial population.

4) The failures of various packages are assumed to be independent of each other.

5) The model is developed for evaluation of the software reliability at the developmental stage and the packages that fail during this period are not further considered. It is further assumed that by the end of developmental stage the bug associated with the failed packages would be eliminated and will be stable further.

6) The reliability of the software is inversely proportional to the number of bugs reported at any point of time.

7) The beginning of the time period after which the bug arrival or failure rate remains constant marks the culmination of the developmental stage and the software will stabilize.

Based on the above assumptions a 6- step algorithm is developed for the analysis as detailed below.

Table 6.1 Algorithm for estimating software reliability

Algorithm. Reliability analysis

Input: **Total Initial Population**

Output: **Expression for Reliability**

Initialize the total population, ***P***

Define a time interval **T<t1, t2>**

for each time interval $\mathbf{T_i}$

Obtain the bugs $b_i \in \{t_1, t_2\}$

Calculate Total failure $T_f = \sum_{i=1}^{n} b_i$

Cumulative failure

$$C_f = number\{k \mid x_k \le t_2\} = \sum_{k=1}^{n} I(x_k \le t_2)$$

where I is the indicator function and $x_1, \ldots\ldots x_n$ are observed data

Survivor $S_i = T_f - C_f^{\,i}$ where $i = 1 to\ 'n' \in T$

Failure rate $F_{r_i} = \dfrac{b_i}{Avg\{C_f(S_i)\}}$

End

Obtain the equation of relation between failure rate and time using regression analysis

Obtain the expression of reliability of the software using $R(t) = e^{-\int_0^t Z(t)dt}$

The algorithm shown above in Table (6.1) is explained as follows

1) Identify the total initial population. This corresponds to the total number of packages existing at the beginning of the time period. That is, at the start of analysis.

2) Define a time period and find out the bugs reported during this time interval. As the failure would have occurred anywhere between the time interval, the reported failures are indicated in between the time interval.

3) Calculate the cumulative failures and thereby the survivors at different points in time.

4) Estimate the failure rate associated with the time intervals by dividing the number of failures associated with the given unit time interval by average population associated with the time interval. Average population associated with a given time interval is the average of survivors at the beginning and end of the time period.

5) Obtain the equation of relation between failure rate and time using regression analysis.

6) Obtain the expression for reliability of the software by substituting the equation of failure rate in equation 6.5 given as

$$R(t) = e^{-\int_0^t z(t)\,dt} \quad \text{---------- (6.8)}$$

6.5 Development of a Simplified Model

The reliability estimation involves considering the computing system as two subsystems, one comprising of hardware components and the other the various software modules or packages. These various packages can be considered as various components of the software part of the system [http://www.debian.org]. These two subsystems are then considered to be connected in series as shown in Fig.6.4.

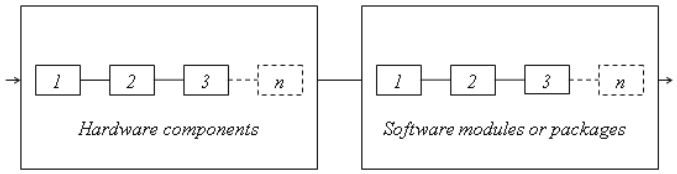


Figure 6.5. Reliability block diagram for the computing system

Considering the entire Reliability Block Diagram (RBD) of the combined system, the reliability of the computing system at time t can be stated as

$$R(t) = R_{h1}.R_{h2}...R_{hn}.R_{s1}.R_{s2}...R_{sn}$$

So, combining the failure rates of both hardware and software the computational system reliability at time *t* can be expressed as:

$$R(t) = R_{hardware} \times R_{software} \quad \text{---------- (6.9)}$$

where, $R_{hardware}$ = Hardware reliability at time *t*

$R_{software}$ = Software reliability at time *t*

Further, if the hardware system components are assumed to have a constant failure rate, the reliability of the hardware part can be expressed as:

$$R_{hardware} = e^{-\sum_{i=1}^{n} \lambda_{h_i} t} \quad \text{----------(6.10)}$$

where, $\lambda_{h_1}, \lambda_{h_2} \ldots\ldots\ldots\ldots \lambda_{h_n}$ represents the failure rate of different hardware components involved in the system.

Equation 6.10 assumes that all the hardware components are essential for the success of the system and as such these components are connected in series in the RBD.

Similarly for software the reliability of the software part can be expressed as

$$R_{software} = e^{-\sum_{i=1}^{n} \lambda_{s_i} t} \quad \text{----------(6.11)}$$

where $\lambda_{s_1}, \lambda_{s_2} \ldots \lambda_{sn}$ represents the failure rate of individual software components involved in the system.

The developed model can be further simplified if the software reliability calculations are based on the Musa model, which assumes a constant failure rate. In this case both hardware and software failure rates are taken to be constant and the computational reliability can be expressed as

$$R(t) = e^{-\sum_{i=1}^{n} \lambda_{h_i} t} \times e^{-\sum_{i=1}^{n} \lambda_{S_i} t} \quad \text{----------(6.12)}$$

where, λ_h represents the failure rate of different hardware components involved in the system and is further expanded as

$$\lambda_h = \lambda_{h1} + \lambda_{h2} + \ldots\ldots\ldots\ldots \lambda_{hn}$$

where $\lambda_{h_1}, \lambda_{h_2} \ldots\ldots\ldots\ldots \lambda_{h_n}$ represent individual hardware components. Similarly $\lambda_s = \lambda_{s_1} + \lambda_{s2} + \ldots + \lambda sn$

where $\lambda_{s_1}, \lambda_{s2} \ldots \lambda_{sn_1}$ represents the failure rate of individual software components.

Now total failure rate of the system is $\lambda_{hardware} + \lambda_{software}$

Ie.

$$R(t) = e^{-(\lambda_{h1} + \lambda_{h2} + \ldots + \lambda_{hm} + \lambda_{s1} + \lambda_{s2} + \ldots + \lambda_{sn})t}$$

$$ie = e^{-\Sigma(\lambda_{hw} + \lambda_{sw})t}$$

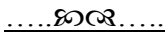
Where λ_{hw} represents the failure rate of hardware components and λ_{sw} represents failure rate of software components. The λ represents components failure rate which in turn is a function of parameters like the measured failure rate of the component (f), the fraction of time spend in the component (t), utilization of CPU by the component (u), and the relative speed of the hardware platform (s). This justifies the fact that

reliability of a computing system depends on both hardware and software components.

The model presented here assume that the components have a constant failure rate. Also the individual components are connected serially. Though the actual relationship among the components in the system are not obvious, the simplified model considers a serial relationship. This could be further investigated, considering the actual relationship among the components. The presented model is to be evaluated against the failure data obtained from various open source software projects. The details are presented in the next chapter.

6.6 Conclusion

The importance of considering both software and hardware together in computational system reliability calculations has been brought out, and a Simplified Model has been developed. The software reliability calculations were based on the developed algorithm whereas the hardware reliability values are obtained using the constant hazard model. The developed algorithm for software reliability estimation can be used as a tool for analysis in open source software development as the necessary input data for the model like bug arrival rate are readily available.



Chapter 7

The Evaluation and Comparison of the Developed Model

Contents

7.1 Introduction

Computation of reliability by integrating software and hardware has got high significance because in any industrial situation these two components should be considered together. The developed simplified model assumes greater importance in this context. The application of the model in industrial situations and subsequent analysis is presented in this section.

7.2 Application of Simplified Model.

A computing system is comprised of two systems in the form of hardware and software. The software part is considered to operate in open source environment. The hardware part of the system is made up of the following components as given in Table 7.1. The respective mean time between failures is also indicated.

Table 7.1 Hardware Component Failures

Sl.No.	Components	MTBF(months)
1	H1	36
2	H2	40
3	H3	35
4	H4	42
5	H5	60
6	H6	24
7	H7	29
8	H8	36

Software reliability is a function of different software packages that are involved in the given system. These various packages are considered as different modules and thus can be considered as various components of the software part of the system. The software component failures are shown in the Table 7.2.

Table 7.2. Software Component Failures

Sl.No.	**Modules**	**Initial MTBF(months)**	**After Development MTBF(months)**
1	M1	18	0
2	M2	18	72
3	M3	16	0
4	M4	18	72
5	M5	17	60
6	M6	16	0
7	M7	13	65
8	M8	14	0
9	M9	18	0
10	M10	17	72
11	M11	16	0
12	M12	15	72

The failure rate of the software during the developmental stage is very high and this is evident from the Fig. 7.1 This means that if the software is used without further modification or development, the reliability values will come down drastically and the software will be impracticable for use, and this demands further development.

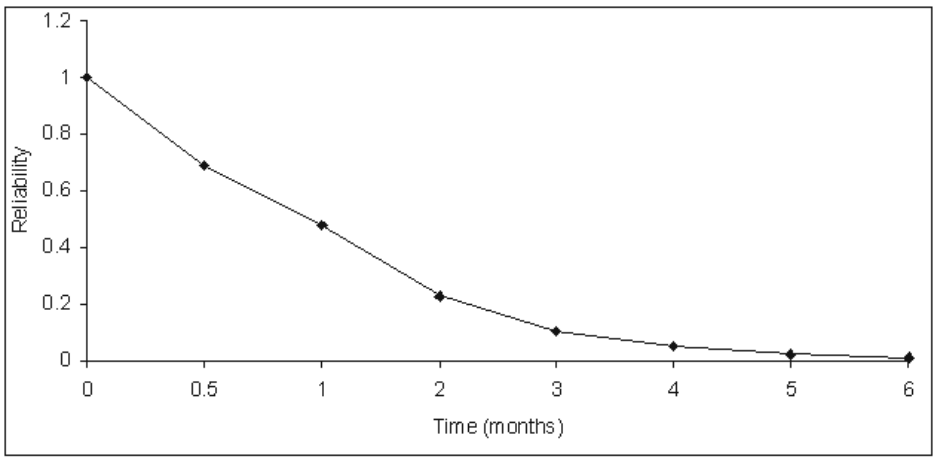


Figure 7.1 Software Reliability Calculated with the Developmental Values of MTBF.

However, after the developmental stage the failure rate comes down drastically and considering that the developmental period is negligibly small in comparison with the life of the system, in order to arrive at the software reliability, the MTBF is based on the failures after the developmental stage. If the reliability calculations are based on the MTBF by considering the failures only after the developmental stage, then the reliability values will be very high and this is evident from the Fig. 7.2.

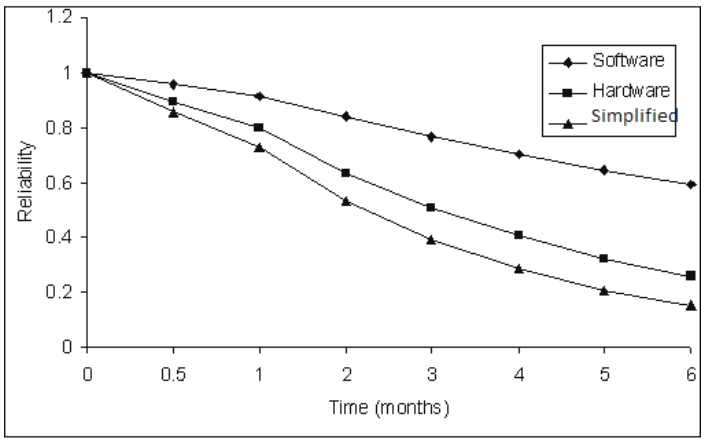


Figure 7.2 Reliability Comparison.

The variation of hardware reliability with time, is also indicated in Fig.7.2. The values of computing system reliability evaluated by considering both hardware and software reliabilities will lie below these two reliability values. This is also evident from Fig. 7.2.

A measure of the error involved in the calculations of reliabilities if the computational reliability calculations are purely based on software part alone, is given in Fig.7.3.

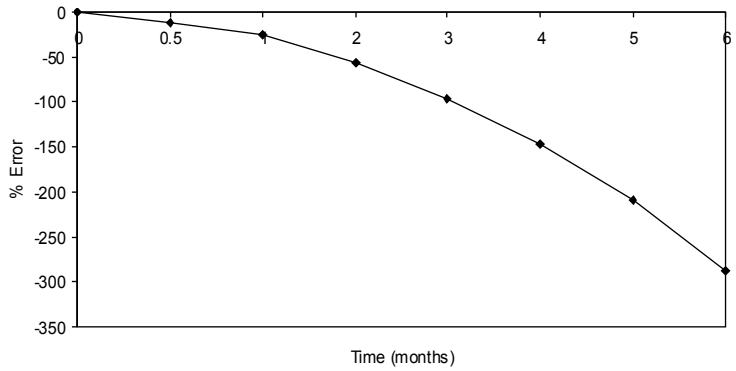


Figure 7.3 Error Involved in Reliability Estimation.

It is evident that if the computational reliability is calculated by considering software and hardware components together, then the total reliability will be much lower than when it is calculated using software alone. The magnitude of error increases with time as the chances of hardware failure is very high with the passage of time.

7.3 Application of Simplified Model in a Real Time Situation

The open source software data is made use of and the methodology for evaluating the software reliability involves identifying a fixed number of packages at the start of the time and defining the failure rate based on the failure data for these preset number of packages. The defined function of the failure rate is used to arrive at the software reliability model. The hardware reliability is obtained using constant hazard model.

A total of 1880 packages were available at the start of the analysis as per the details available from the official website of Debain. This is taken as the initial population. A time interval of 1 month is fixed and the bug arrival rate during this interval is noted. The reported errors at different time intervals are given in the Table 7.2. The observations are taken for 1 year after which, the bug arrival is negligible indicating that the software has more or less stabilized.

Table 7.3 Software Failure Data Analysis

Time	No. of Failures	Cumulative Failures	Survivors	Failure Rate (per month)
Feb-08		0	1880	
	25			0.013386881
Mar-08		25	1855	
	61			0.033433817
April-08		86	1794	
	340			0.209350606
May-08		426	1454	
	49			0.034277719
June-08		475	1405	
	55			0.039927405
Jul-08		530	1350	
	214			0.172164119
Aug-08		744	1136	
	136			0.127340824
Sept-08		880	1000	
	37			0.037697402
Oct-08		917	963	
	40			0.042417815
Nov-08		957	923	
	48			0.048929664
Dec-08		1005	875	
Average				0.075893525

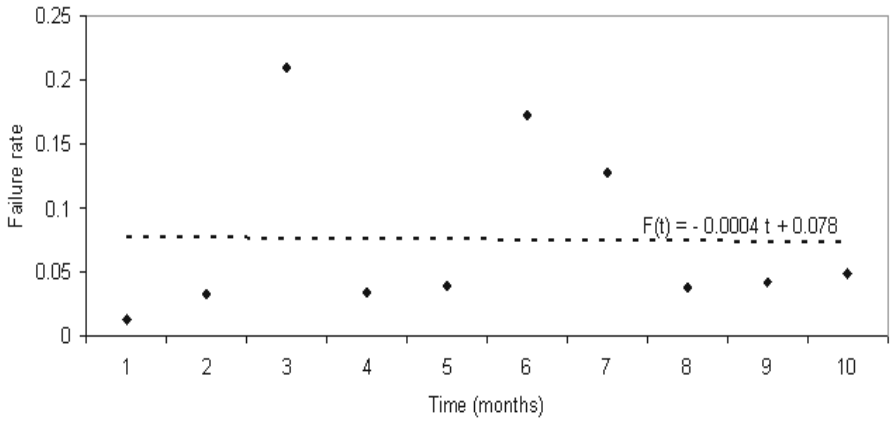


Figure 7.4 Software Failure Rate

The variation of software failure rate with respect to time is shown in Fig. 7.4. It can be seen that after the 8th month onwards the software has somewhat stabilized indicating the completion of developmental phase. The failure model corresponding to the failure rate can be expressed as:

$$Z(t) = -at + b \quad \text{---------- (7.1)}$$

Where a=0.0004 and b = 0.078.

The corresponding reliability can be expressed as:

$$R_{Software} = e^{-\int_0^t (-0.0004t + 0.078)dt}$$

$$= e^{\frac{0.0004t^2}{2} - 0.078t} \quad \text{---------- (7.2)}$$

Table 7.4 Hardware Component Failure Rate

Hardware Component No.	Failure Rate (per month)
H_1	0.027778
H_2	0.025
H_3	0.028571
H_4	0.02381
H_5	0.016667
H_6	0.041667
H_7	0.034483
H_8	0.027778

The failure rate of the hardware components are indicated in Table 7.4. The values of the hardware failure rate can be substituted in equation 7.2 to get the hardware reliability equation and can be expressed as:

$$R_{hardware} = e^{-\sum_{i=1}^{n} \lambda_{h_i} t}$$

$$= e^{-0.225753t} \quad \text{---------- (7.3)}$$

Thus, the reliability of the computing system R(t) at any given time will be the product of equations 7.2 and 7.3 and can be expressed as:

$$R(t) = e^{\frac{0.0004t^2}{2} - 0.078t} \times e^{-0.225753t}$$

$$= e^{\frac{0.0004t^2}{2} - 0.303753t} \quad \text{---------- (7.4)}$$

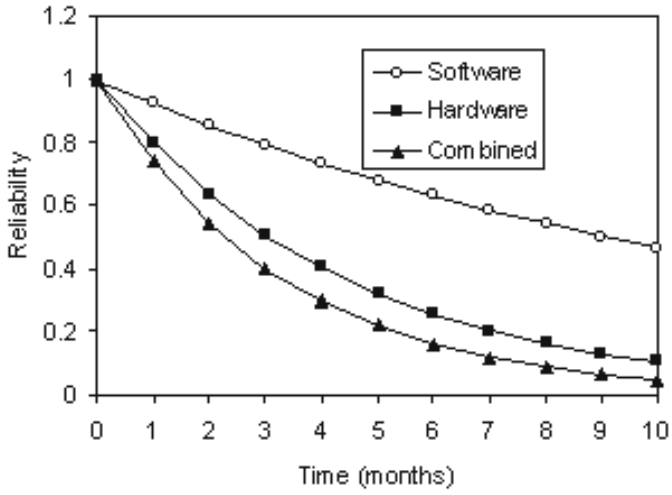


Figure 7.5. Variation of Reliability with Time

Fig. 7.5 shows the variation of reliability with respect to time when software and hardware are considered independently also the combined reliability using the developed simplified model.

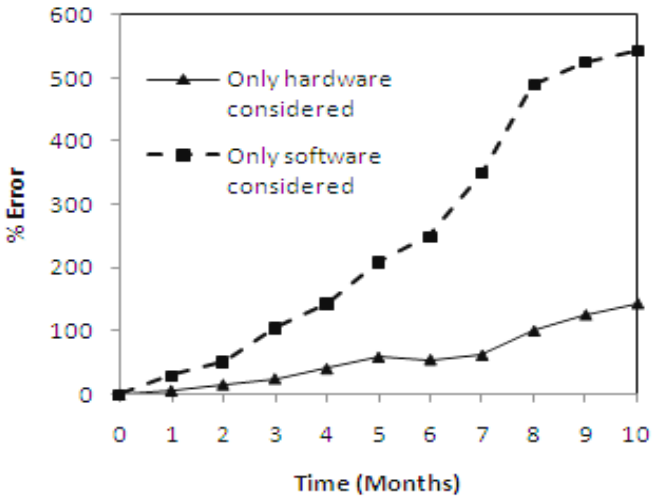


Figure 7.6. Error Involved in Computational Reliability Calculations

Fig 7.6 shows the error involved in computational reliability calculations when hardware or software components alone are considered. It is evident that the error involved is quite significant and thus necessitates the incorporation of both these elements.

The reliability of the software at different points in time is calculated using the equation (7.2). The actual values of reliability obtained by dividing the survivors at the given point in time by the initial population are also calculated. The Musa model assumes a constant value for the failure rate and by considering this as the average value of failure rates the reliability values are calculated using the equation

$$R(t) = e^{-\lambda t} \quad \text{----------} \quad (7.5)$$

The reliability values for Weibull distribution is calculated using the equation

$$R(t) = e^{-(t/\alpha)^{\beta}} \quad \text{----------} \quad (7.6)$$

The reliability values calculated using the four different methods and the failure density values are shown in Table (7.5).

Table 7.5. Reliability and Failure Density

Time	Failure Density	Reliability (Actual)	Reliability (Musa)	Reliability (Weibull)	Reliability (Model)
08- Feb		1.00000	1.00000	1.00000	1.00000
	0.013297872				
08- Mar		0.98670	0.85985	0.98144	0.92515
	0.032446809				
08- April		0.95426	0.73934	0.92263	0.85624
	0.180851064				
08- May		0.77340	0.63572	0.82780	0.79279
	0.02606383				
08- June		0.74734	0.54662	0.70739	0.73433
	0.029255319				
08- Jul		0.71809	0.47001	0.57487	0.68045
	0.113829787				
08- Aug		0.60426	0.40414	0.44377	0.63078
	0.072340426				
08- Sept		0.53191	0.34750	0.32507	0.58497
	0.019680851				
08- Oct		0.51223	0.29879	0.22577	0.54270
	0.021276596				
08- Nov		0.49096	0.25692	0.14856	0.50369
	0.025531915				
08- Dec		0.46543	0.22091	0.09256	0.46767

Fig.7.7 shows a comparison of reliability obtained using the Developed, Simplified model, and Weibull and Musa model with the actual reliability values. It can be seen that the simplified model shows a better result comparing to other models. Further, these two models very closely approximate the real situation.

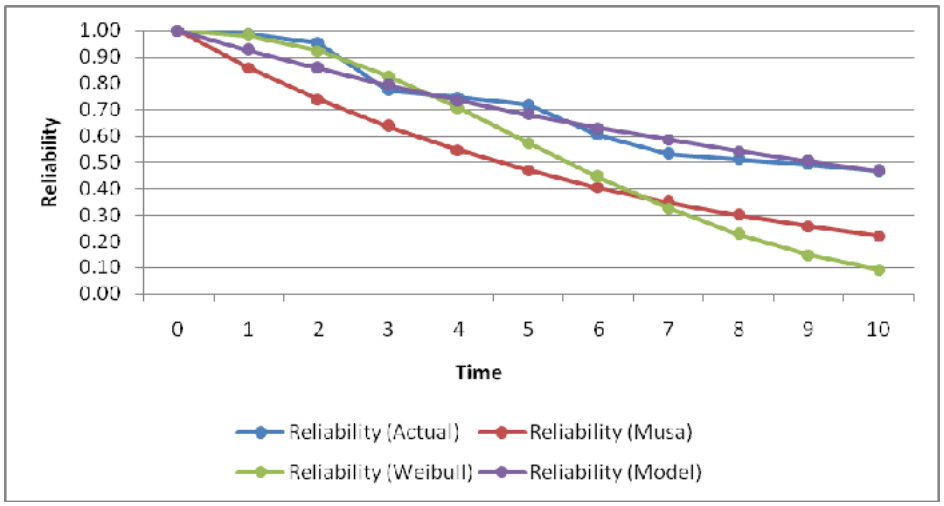


Figure 7.7. Comparison of Reliability Obtained Using Different Models

The variation of failure density with time is also shown in Fig.7.8. It can be seen that the failure density increases in the initial stage and comes to a peak value, then decreases again increases and finally comes to a stable state at the end.

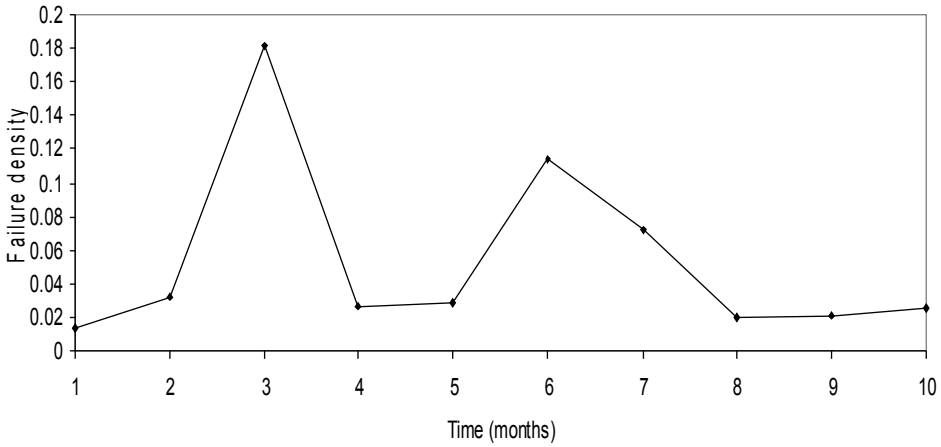


Figure 7.8 Variation of Failure Density with Time

Fig 7.9 compares the reliability value obtained using the model with the theoretical value. It can be seen that the percentage error is always within 10% of the actual value which is a reasonably a good result for all engineering problems.

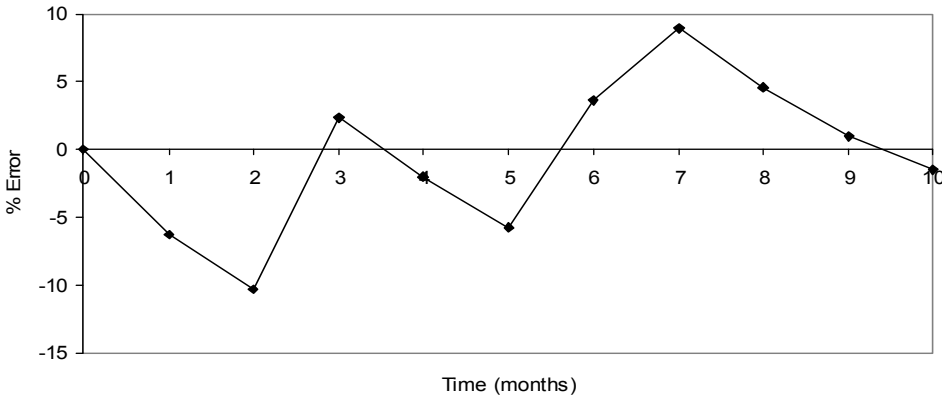


Figure 7.9. Error Analysis

7.4 Conclusions

A new method for estimation of reliability of computational systems was developed. The methodology is far more realistic in comparison with the traditional methods which focuses either on hardware or software alone, rather than integrating these elements. The concept is very much in accordance with the systems approach. The developed method could prove to be a very effective tool for reliability analysis of computational systems. An error analysis was also conducted and it can be seen that if the hardware and software components are not integrated in reliability analysis the calculated values will be an over estimated one. That is, the calculated values would be much higher than

the actual reliability values. A comparison of reliability obtained using the developed model, Weibull and Musa model with the actual reliability values are also shown.

.....ജോ.....

Chapter 8

Research Findings and Conclusions

Contents

8.1 Introduction

The major objective of the research was to develop an integrated model for estimation of computational system reliability by combining both hardware and software. The hardware system reliability was evaluated using a Constant Hazard Model and the software reliability was obtained using the Musa Model, Weibull Model as well as the developed model. The research findings emphasize the importance of incorporating both hardware and software in computational reliability calculations.

8.2 Research Findings and Outcome

The major findings of the research work are listed below:

An exhaustive survey of existing reliability growth models for software, hardware and open source software were carried out and a taxonomy of reliability models were presented.

A frame work for early prediction of software reliability consisting all phases of software process was proposed.

A case study conducted revealed the trend towards adoption of open source by various organizations including government. The study also brought out the importance of thorough analysis and evaluation of the product before the adoption.

Reliability analysis of a handful number of open source software projects were carried out. An attempt made to fit a reliability model for these selected projects, reveal Weibull distribution as a candidate.

The importance of development of an integrated model for prediction of overall reliability of a software system was brought out. An algorithm for software reliability calculation is presented and a new model incorporating hardware reliability is proposed.

The proposed model was validated with real data and the usefulness was demonstrated comparing the model with existing reliability models. An error analysis was also carried out to prove the effectiveness of the integrated approach..

8.3 Research Contributions

The contribution of this thesis is twofold. First a comprehensive study on reliability of FOSS and an investigation on incorporating reliability in each stages of software development life cycle has been carried out. This aspect is important since the development model

followed by FOSS and closed source software are entirely different. Secondly the proposed model helps to enable industry to zero-in appropriate FOSS products. The details of contributions towards researchers and practitioners community are discussed below.

8.3.1 Contributions towards Practitioners

Software companies build quality products by following key practice areas proposed by models like capability maturity model (CMMI). These models are not directly useful in the case of open source domain. The thesis throws light into how to fuse quality in an open source project, by proposing reliability measures in each stage of the software process model. The proposed model is helpful to the practitioners to judge the suitability of open source software in terms of reliability and adaptability, before the integration with the existing environment.

8.3.2 Contributions towards Researchers

A reliability model for a computational system by integrating hardware and software component reliability is developed and there is no such model found in literature. The developed model is validated with experimental data pertaining to OSS. We have also compared the model with existing ones and suggest this model for reliability computation of OSS. Most of the work from the thesis were published in international and national journals and international and national conferences.

8.4 Limitations and Further Scope

The present study is an attempt to develop an integrated model for assessing reliability of software system. The study focused on data available from open source software and related hardware failure. The work considered data obtained from major open source software projects for the analysis and development of the model. This could be further improved if data from more projects were available but many of the projects were stopped and abandoned, and collection of data was difficult. It was also difficult to obtain data pertaining to hardware failures.

The evaluation of the developed model was done against two well known models. More models could have been included in the study but well accepted models were not available.

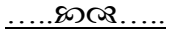
Even though many projects are available in the open source software domain successful projects were limited in number. The developed model could be further toned to its perfection if more and more data from successful projects can be incorporated.

An automated tool for reliability analysis of open source software projects does not exist. This can be developed as a further work.

8.5 Conclusion

The importance of considering the software and hardware together in computational reliability calculations is the theme of thesis. A simplified model incorporating both these elements was developed and

applied in real time situations. The impact of neglecting any one of these elements, that is hardware or software, is brought out, and more importantly, the error involved will be enormously high when the hardware part is not considered. The developed model can prove to be a very useful tool in reliability analysis of computing systems.



References

[1]. A. Mockus, T.R. Fielding, and J.D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla", ACM Transactions on Software Engineering and Methodology, vol. 11, no. 3, July 2002, pp. 309-346.

[2]. A.L. Goel and K. Okumoto, "A time-dependent error-detection rate model for software reliability and other performance measure", IEEE Transactions on Reliability, vol. R-28, 1979, pp. 206-211.

[3]. Alan M. Davis. Great Software Debates (October 8, 2004), pp:125-128 Wiley-IEEE Computer Society Press.

[4]. Allen Nikora and Michael Lyu, "Software Reliability and Risk Management: Techniques and Tools", tutorial presented at the 1999 International Symposium on Software Reliability Engineering.

[5]. Allen Nikora, John Munson, "Determining Fault Insertion Rates For Evolving Software Systems", proceedings of the International Symposium on Software Reliability Engineering, Paderborn, Germany, November, 1998.

[6]. Amitabha Yadav and R.A. Khan "Reliability Estimation Framework-Complexity perspective" ICAITA, SAI, SEAS, CDKP, CMCA, CS & IT 08, pp. 97–104, 2012. © CS & IT-CSCP 2012.

[7]. Amrit L and Goel. Software reliability Models: Assumptions, Limitations, and Applicability. *Transactions on Software Engineering*, Vol. 2, No. 12, pp 1411-1423, 1985 IEEE.

[8]. Ashis Arora, V.S. Arunachalam, Jai Asundi, Ronald Fernandes "The Indian Software Industry".

[9]. B. Bream, Curator, "Reliability Block diagrams and Reliability Modeling" , Office of Safety and Mission Assurance, NASA Lewis Research centre, may 1995.

[10]. B. Littlewood and J.L. Verrall, "A bayesian reliability growth model for computer software", Applied Statistics, vol. 22, 1973, pp. 332-346.

[11]. B. Littlewood and J.L. Verrall, "A bayesian reliability model with a stochastically monotone failure rate", IEEE Transactions on Reliability, vol. R-23, June 1974, pp. 108-114.

[12]. Bain, I. J. (1974) Analysis of linear failure rate life testing distribution. Technometrics 16:551-60

[13]. Bajaj . A(2000) A study of senior IS managers decision model in adopting new computing architecture, JAIS, vol. 1, Paper4, pp 1-58.

[14]. Boyd M A and Monahan C M. Developing Integrated Hardware - Software reliability Models: Difficulties and Issues. Digital Avionics Systems Conference, pp 193-198, 1995 IEEE.

[15]. Bugzilla, http://www.bugzilla.org.

[16]. Capiluppi A and Michlmayr M 2007, in IFIP International Federation for Information Processing, volume 234, Open source development , Adoption and innovation,eds. J Feller, Fitzgerald, B, Scacchi, W, Sillitti, A(Boston Springer) pp 31-44.

[17]. Carolyn A. Kenwood A business case study of open source software 2001 The MITRE Corporation.

[18]. Charles EE. Reliability and maintainability engineering.1st ed. New Delhi: Tata McGraw-Hill Publishing Company Ltd; 2000.

[19]. Charles Jobson "J2EE Design in UML using RUP and Agile Software Development"Callista Enterprise Developer's Conference 29 Jan 2003.

[20]. Chau.P and Tam.K(1997) Factors affecting the adoption of open systems:an exploratory study, MIS quarterly, Vol.21.No.1. pp. 1-24.

[21]. Clark T., P. Sammut & J. Willans, "Applied Metamodelling: A Foundation for Language Driven Development", Second Edition, Ceteva 2008.

[22]. Cleland D.L & King W.R: System Analysis and Project Management, Mcgraw Hill.1992.

[23]. Cobra Rahmani, Harvey Siy and Azadmanesh "An Experimental Analysis of Open Source Software Reliability" http://www.cse.buffalo.edu/srds2009.

[24]. Cooper. R and Zmud, R.(1986) Information Technology implementation: a technological diffusion approach, Management Science, vol 36, no.2,pp156-172.

[25]. Coutinho J. de S, "Software Reliability Growth ,"IEEE Symposium on Computer Software Reliability, 1973.

[26]. Debra .s and David .E." Software Reliability Cases: The Bridge between hardware, software and system safety and reliability" proceedings Annual Reliability and Maintainability symposium, 1999.

[27]. Dhillon, B. S. (1981) Life distributions. IEEE Transactions on Reliability R30:457-9.

[28]. Dominik Richter, Hangjung Zo, Michael Maruschke, "A comparative analysis of open source software usage in germany, brazil, and india", 2009 fortth International conference on computer science and convergence Information Technology, 2009 IEEE, pp 1403-1410.

[29]. DRM Associates (2002). "New Product Development Glossary". http://www.npd-solutions.com/glossary.html. Retrieved 2006-10-29.

[30]. Dzumbu M. Edge Analyst at AEL Mining Services 2008.

[31]. E.S. Raymond, "The cathedral and the bazaar: musings on linux and open source by an accidental revolutionary, 2nd Ed., O'Reilly, 2001.

[32]. Edelstein, K. L. (1998) Mathematical models in biology. Random House, New York.

[33]. El-Emam. Ethics and Open source. Empirical Software Engineering 4, 6 (2001), 291-292

[34]. Eugene Glynn, Brian Fitzgerald and Chris Exton, "Commercial adoption of Open Source Software: An Emperical Study", 2005 IEEE.

[35]. Eveland, J and Tornatzky.L(1990) The deployment of technology, in Tornatzky, L and Fleischer, M(eds) The process of technological innovation, Lexington Books.

[36]. Federic P, Brooks J. R *Chap. 17*. "'No Silver Bullet' Refired". *The Mythical Man Month* (Anniversary Edition with four new chapters ed.) (Addison-Wesley). ISBN 0-201-83595-9, (1995).

[37]. France R. B. and B. Rumpe, "Model-driven development of complex software: A research roadmap," in FOSE 07: 2007 Future of Software Engineering, Washington, DC, USA:IEEE Computer Scociety, 2007, pp. 37-54.

[38]. Franz Brosch, Heiko Koziolek "Architecture-Based Reliability Prediction with the Palladio Component Model" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 38, NO. 6, NOVEMBER/DECEMBER 2012.

[39]. Gacek, C. and Arief, B., The many meanings of open source, IEEE Software, Vol. 21, No. 1, pp. 34-40, 2004.

[40]. Gaver, D. P. and M. Acar. (1979) Analytical hazard representation for use in reliability, mortality and simulation studies. Communications in Statistics-Simulation and Computation B8 (2):91-111.

[41]. Goel .A.L, "Software reliability models: Assumptions,limitations, and applicability", *IEEE Trans. Software Engineering,*vol SE-11, num 12, 1985, pp 1411 - 1423.

[42]. Goel A.L. and K.Okumoto, "Time-Dependent Error-Detection Rate Model for software and other performance measures," IEEE Trans. Reliability, vol. 28, pp. 206-211, 1979.

[43]. Goel HD, Grievink J, Herder PM, Weijnen MPC. Integrating reliability optimization into chemical process synthesis. J Reliability Engineering and System Safety 2002; 78:247–258.

[44]. Goseva-Popstojanova K. et al. Architectural Level Risk Analysis using UML, *IEEE Transactions on Software Engineering*,Vol.29, No.10, October 2003.

[45]. Greenfield J. , K. Short. S.Cook. and S. Kent, Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, Indianapolis, 2004.

[46]. H. Pham, Software Reliability. Springer-Verlag, 2000.

[47]. H.S. Kan, Metrics and Models in Software Quality Engineering, 2ndEd., Addison-Wesley, 2003.

[48]. Haupt, E. and H. Schabe. (1992) A new model for life time distribution with bathtub shaped failure rate. Microelectronics Reliability 32(5):633-9.

[49]. Hijroth, U. (1980) A reliability distribution with increasing, decreasing, constant and bath-tub shaped failure rates. Technometrics 22(1):99-107.

[50]. Hoang Pham, Xuemei Zhang, "A Software Cost Model with Warranty and Risk Costs,"IEEE Transactions on Computers, vol. 48, No.1, January 1999.

[51]. Hossain S.A and R.C.Dhahiya, "Estimating the parameters of a Non-Homogeneous Poisson Process Model for software Reliability Model for Software Reliability," IEEE Trans. Reliability, vol.42, pp. 604-612, 1993.

[52]. Hsieh, Y.C., T. C. Chen and D. L. Bricker. (1998) Genetic algorithm for reliability design problems. Microelectronics Reliability 38:1599–605.

[53]. http://www.coverity.com

[54]. http://www.debian.org.

[55]. http://www.gnu.org/philosophy/philosophy.html

[56]. http://www.govtalk.gov.uk/policydocs

[57]. http://www.mit.gov.in

[58]. http://www.oscc.org.my/content/view/227/139/

[59]. http://www.pcguide.com/care/bu/risksHardware-c.html

[60]. http://xface.itc.it.

[61]. http:/www.opensource.org/docs/definition.php

[62]. Huang C.Y, M. R. Lyu and S. Y. Kuo "A unified scheme of some non-homogenous Poisson process models for software reliability estimation" *IEEE Trans. Softw. Engineering, vol. 29, no. 3, pp. 261-269, 2003*

[63]. I. Koren and C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufmann, 2007.

[64]. IAN SOMMERVILLE, Software Engineering. Pearson Education publishing, 2003.

[65]. IEEE Reliability Society, "IEEE recommended practice on software reliability", IEEE Std 1633-2008, June 2008.

[66]. IEEE Std 1413-2010, IEEE Standard Framework for Reliability Prediction of Hardware.

[67]. IEEE/AIAA P163/DRAFT 14, Reccommended Practice on Software Reliability[2007].

[68]. "IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software", 1998

[69]. J.D. Musa and A. Iannino. Software reliability modelling-accounting for program size variation due to integration or design changes. Proceedings of the 1981 ACM workshop/symposium on Measurement and evaluation of software quality, 1981. pp 129-130

[70]. J.D. Musa and K. Okumoto, "A logarithmic poisson execution time model for software reliability measurement", 7th Int'l Conference on Software Engineering (ICSE), 1984, pp. 230-238.

[71]. J.De.S Coutinho, "Software reliability growth". IEEE Symposium on Computer Software Reliability, 1973, pp. 58-64.

[72]. Jagadeesh Nandigam, Venkat N Gudivada and Abdelwahab Hamaou-Lhadj, "Learning Software Engineering Principles using Open Source Software", Proceedings of the 38th ASEE/IEEE Frontiers in Education Conference,2008.

[73]. Jaisingh, L. R., W. J. Kolarik and D. K. Dey. (1987) A flexible bathtub hazard model for non-repairable systems with uncensored data. Microelectronics reliability 27(1): 87-103.

[74]. Jean Dolbec and Terry Shepard 1995, "a component based software reliability model".

[75]. Jeff Norris Mission- critical Development with Open Source Software: Lessons lerned. IEE software.

[76]. Jelinski Z. and P. Moranda, Software reliability research", In Statistical Computer performance evaluation , W. Frieberger, Ed., New York: Academic, 1972, pp. 465-484.

[77]. Joe Palma, Jeff Tian and Peng L u. Collecting Data for Software Reliability Analysis and Modeling. *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering,* Vol1, 1993, pp 483-494

[78]. Joseph p, james .a " A frame work for the measurement of software quality" proceedings of the software quality assurance work shop on functional and performance issues, p 133-139, 1978.

[79]. K. Neubeck, Practical Reliability Analysis, Prentice Hall, 2004.

[80]. K.S. Wang, F.S. Hsu and P.P. Liu "Modeling the bathtub shape hazard rate function in terms of reliability", Reliability Engineering & System Safety Volume 75, Issue 3, March 2002, Pages 397–406.

[81]. Kalpana Yadav ,Eshna Jain, Jaspreet Bhatia " Comparative Study of Open Source Software Reliability Assessment Models",2011 IEEE..

[82]. Kan H.S, Mmetrics and Models in software Quality Engineering, 2nd Ed. Addison-Wesley, 2003.

[83]. Kan H.S. Metrics and models in software quality engineering, 2nd edition, Addison-Wesley (2003)

[84]. Kemal Isitan, Timo Nummenmaa and Elini Berky. Openness as a method for game evolution, 5 pages. To appear in proceedings of the IADIS Inter National Conference in Game and Entertainment Technologies 2011.

[85]. Kenneth Wong, "Free/Open Source Software: Government Policy", International open source network, Asia-Pacific Development Information Programme, e-Primers on Free/Open Source Software , 2004.

[86]. Kevin Crowston and Barbara Scozzi. Exploring the strengths and limits of open source software engineering processes: A research agenda. Second Workshop on open Source Software Engineering, Orlando, Florida. May 25, 2002.

[87]. Kevin Lewis "Eight reasons why software implementation projects fail" www.powerandgasmarketing.com • Spring 2003.

[88]. Kumar D, Klefsjo B, Kumar U. Reliability analysis of power-transmission cables of electric loaders using a proportional-hazard model. J Reliability engineering & system safety 1992; 37:217–22.

[89]. Kuo, W. and V. R. Prasad. (2000) An annotated overview of system-reliability optimization. IEEE Transactions on Reliability 49(2): 176–87.

[90]. Kuo, W., V. R. Prasad, F.A. Tillman and C. Hwang. (2001) Optimal reliability design. Cambridge University Press, Cambridge.

[91]. Lakey, Peter and Neufelder, Ann Marie, "System and Software Reliability Assurance Notebook," Rome Laboratory Report, Griffiss Air Force Base, Rome NY, 1997. http://www.cs.colostate.edu/~cs530/rh/

[92]. Laprie and Karama Kanoun, LAAS-CNRS, Toulouse, France, "Software Reliability and System Reliability".

[93]. Laprie, J-C and Kanoun, K, "X-Ware Reliability and availability modeling", IEEE Transactions on Software Engineering vol.18, no.2,February 1992, pp,134-139.

[94]. Lawless J.F. Statistical Models and Methods for Lifetime Data, 2nd edition, New York: Wiley (2003)

[95]. Lawless, J. F. (1982) Statistical models and methods for life time data. Wiley, New York.

[96]. Leblanc S. P. , P.A.Roman, "Reliability Estimation of Hierarchical Software Systems.", 2002 Proceedings annual reliability and maintainability symposium.

[97]. Leslie Cheung, Roshanak Roshandel, Nenad Medvidovic, Leana Golubchik ",Early Prediction of Software Component Reliability" *ICSE' 08*, May 10 - 18, 2008, Leipzig, Germany.Copyright 2008 ACM 978-1-60558-079-1/08/05...$5.00.

[98]. Leung Y.W., "Optimal Software Release Time with a given Cost Budget," J. Systems and Software, vol. 17. Pp. 23-242, 1992.

[99]. Li M. and C.Smidts,' A Ranking of Software Engineering Measures based on Expert Opinion," IEEE Transactions on Software Engineering, vol. 29,pp, 811-24, 2003.

[100]. Li M., Y.Wei, D.Desovski, H.Nejad, S.Ghose, B Cukic, C. Smidts., "Validation of a methodology for Assessing Software Reliability" proceedings of the 15th international symposium on Software reliability Engineering (ISSRE '04)1071-9458/04.

[101]. Littlewood B. and J.L. Verrall, A Bayesian Reliability Growth Model for Computer Software, J. Royal Statist. Soc., C (Applied Statistics), Vol. 2, pp 332-346, 1973.

[102]. Lloyed D. K. and M. Lipow. 1977. Reliability: management, methods, and mathematics, 2nd ed. Redondo Beach, CANSWER: published by the authors.

[103]. Luyin Zhao and Sebastian Elbaum A Survey On Quality Related Activities in Open Source**;** Software Engineering Notes Vol 25 no 3: 54-57 May 2000.

[104]. Lyu M.R "Design, Testing, and Evaluation Techniques for Software Reliability Engineering**"**1998.

[105]. Lyu M.R (ed.*), Handbook of Software Reliability Engineering*, McGraw-Hill and IEEE Computer Society Press, New York, 1996.

[106]. M.L. Shooman, "Probabilistic models for software reliability prediction", in Statistical Computer Performance Evaluation, W. Freidberger, Ed., New York: Academic Press, 1972, pp. 485-502.

[107]. Mark Sondheim Open Source Geospatial Software. The Jump-Project, March 2004.

[108]. MartinMichlmayr, FrancisHunt, DavidProbert Quality Practices and Problems in Free Software Projects, Proceedings of the First International Conference on Open Source Systems Genova, pp 24-28, July 2005

[109]. Michael R Lyu Software Reliability Engineering: A Roadmap, Future of Software engineering (FOSE 07) May 2007 IEEE.

[110]. Michael R. Lyu, *Handbook of Software Reliability Engineering*. McGraw-Hill publishing, 1995.

[111]. Mike. H. "Data Management and Systems Operations at Net Edge Solutions" 2008

[112]. Ming-Wei Wu, Ying Dar – Lin, Open Source Software Development: An Overview. Computer, 34(6) 33 - 38, June 2001.

[113]. Moranda P.B, "Software Reliability Predictions" Proceedings of the Sixth Triennia World Congress of the International Federation of Automatic Control, 1975, pp 342-347.

[114]. Moranda.P.L. and Jelinski. Z.,Final Report on Software Reliability Study, McDonnell Douglas Astronautics Company, MADC Report Number 63921,1972.

[115]. Musa J. D. 2005 "software reliability engineering" tata McGraw – Hill Edition 2005.

[116]. Musa J.D "Meassurement and Management of Software Reliability" proceedings of the IEEE, VOL. 68, NO. 9, September 1980.

[117]. Musa J.D, A. Iannino, K. Okumoto, *Software Reliability,*1987; McGraw-Hill.

[118]. Musa J.D. and K. Okumoto, A Logarithamic Execution time model for software reliability measurement, Proc. 7th International conference on Software Engg., Orlando, Florida, march 26-29, pp 230-238, 1984.

[119]. Musa J.D. and A. Iannino and K. Okumoto, Software Reliability, Measurement, Prediction, Application, McGraw-Hill, 1987.

[120]. Musa J.D. and Okumoto K. "A Logarithamic Poisson Execution Time Model for Software Relibility Meassurement,"Proceedings Seventh International Conference on Software Engineering, Orlando, Florida,1983, PP.230-238.

[121]. Musa J.D., "A Theory of Software Reliability and Its Applications", September 1975, IEEE Transactions on Software Engineering, Volume: SE-1, No: 3, pp.312-327

[122]. Musa. J. D. and A. Iannino. Software reliability modelling-accounting for program size variation due to integration or design changes. *Proceedings of the 1981 ACM workshop/symposium on Measurement and evaluation of software quality*, 1981. pp 129-130

[123]. Myron Hecht, Karen Owens, Joanne Tagami "Reliability-Related Requirements in Software-Intensive Systems" 2007 IEEE.

[124]. Nahid golafshani(2003) 'Understanding reliability and validity in qualitative research' the qualitative report voleume 8 number 4 december 2003 597-607. http://www.nova.edu/sss/QR/QR8-4/golafshani.pdf.

[125]. Norman Schneidewind,'Tutorial on Hardware and Software Reliability, Maintainability, and Availability" 2008,IEEE.

[126]. Octavio B. Management Consultant in Business Analytics & Optimization at IBM 2008

[127]. Ohba M., "Software Reliability Analysis Models," IBM J. Research and Development, vol 28, pp. 428-443, 1984.

[128]. Open Source Software development – Wikipedia, the free encyclopedia-. *en.wikipedia.org*.

[129]. Pankaj Jalote, and Rajib Ghosh, "An Approach for Cost Effectiveness Analysis of Multiversion Software Using Software Reliability Models" http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.3640.,2011

[130]. Patrick DTO. Practical reliability engineering, 4th ed. England: John Wiley & Sons Ltd; 2002.

[131]. Pekka Aho, Janne Merilinna, Eila Ovaska, "Model-Driven Open Source Software Development", 2009 Fourth International Conference on Software engineering Advances, 2009IEEE.

[132]. Pham H "Software reliability and cost models: perspectives, comparison and practice" *European J. of Operational Research, vol. 149, pp. 475-489, 2003*

[133]. Pham H, "Software Reliability Assessment: Imperfect Debugging and Multiple Failure Types in Software Development,"EG&G-RAAM-10737, Idaho Nat'1 Eng. Laboratory,1993.

[134]. Pham H. "System Software Reliability" http://www.springer.com, , XIV,440 p.63 illus., Hardcover ISBN, 2007.

[135]. Popstajanova K. and K. Trivedi "Architecture based approach to reliability assessment of software systems" *Performance Evaluation, vol. 45, no. 2, 2001*

[136]. R Project, http://www.r-project.org/.

[137]. R.C. Cheung, "A user-oriented software reliability model", IEEE Transactions on Software Engineering, vol. 6, no. 2, March 1980, pp. 118-125.

[138]. Report of FOSS usage in the US Department of Defence, prepared by MITRE Corporation: www.egovos.org/ awmedia_repository/ 588347ad_c97c_48b9_a63d_821cb0e8422d?/document.pdf

[139]. Reussner R. et al. Reliability prediction for component-based software architectures, *J. Systems and Software, 66(3),* 2003.

[140]. Richard unkle, Ray Venkataraman. Quality & reliability corner relationship between Weibull and AMSAA models in reliability analysis a case study. International Journal of Quality & Reliability Management, vol.19, pp. 986-997, 2002.

[141]. ROGER S PRESSMAN,*Software Engineering*. McGraw-Hill publishing, 2007.

[142]. Rosenberg L., T. Hammer, J. Shaw, "Software Metrics and Reliability", 1998. http://satc.gsfc.nasa.gov/support/ISSRE_NOV98/ software_metrics_and_reliabil ity.html.

[143]. S. Yacoub, B. Cukic and H.H. Ammar, "A software-based reliability analysis approach for component-based software", IEEE Transactions on Reliability, vol. 53, no. 4, Dec 2004.

[144]. S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection", IEEE Transactions on Reliability, Vol. R-32, 1983, pp. 475-478.

[145]. S.S. Gokhale, M.R. Lyu, and K.S. Trivedi, "Reliability simulation of component-based software systems", Proceedings of 9th Int'l Symposium on Software Reliability Engineering, 1998.

[146]. Schabe Hendrik. (1994) Constructing lifetime distributions with bathtub shaped 87 failure rate from DFR distributions. Microelectronics and Reliability. 34(9):1501-8.

[147]. Scott Christley and Greg Madey "Analysis of Activity in the Open Source Software Development Community" Proceedings of the 40th Hawaii International Conference on System Sciences-2007 IEEE.

[148]. Sergio Bittanti, An Introduction to Software Reliability Modeling, Lecture notes in Computer Science,341, Sergio Bittanti(ED.), Springer-Verlag, 1988.

[149]. Sharma, K. Garg, R. Nagpal, C.K. Garg, R.K. "Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach" Reliability, IEEE Transactions on, Issue Date: June 2010.

[150]. Shiyi Xu. "An Accurate Model of Software Reliability' 13th IEEE International Symposium on Pacific Rim Dependable Computing, 2007 IEEE.

[151]. Shooman M.L "Structural Models for Software reliability Prediction," Proceedings of the 2nd International Conference on Software Engineering, IEEE, Computer Society, New York, October 1976.

[152]. Shooman M.L 1986. Probabilistic reality: an engineering approach. 1968. New York: McGraw- Hill. Updated and reprinted, Kreger, Malabar, FL. 1986.

[153]. Shooman M.L, "Probablistic Methods For Software Reliability Prediction", Statistical Computer Performance Evaluation, Academic Press, New York, June 1972, pp 485-502.

[154]. Shooman M.L,"Operational Testing and Software Reliability Estimation During Program Developments," Record of 1973 IEEE Symposium on Computer Software Reliability, IEEE Computer Society, New York, 1973, pp. 51-57.

[155]. Shooman M.L. "Spectre of Software Reliability and its Exorcism," Proceedings of the 1977 Joint Automatic Control Conference , IEEE, New York, 1977, pp-225-231.

[156]. Shooman, M. I. (1968) Probabilistic reliability: an engineering approach. McGraw Hill, New York.

[157]. Shouri P.V,P.S. Sreejith Algoritham for break even availability allocation in process system modification using deterministic valuation model incorporating reliability jan 2008 ELSVIER ScienceDirect.

[158]. Smidts C., R. W. Stoddard and M. Stutzke "Software reliability models: An approach to early reliability prediction" *IEEE Trans. Reliability, vol. 47, no. 3, pp. 268-278, 1998*

[159]. Smidts C , B.Li, and Z.Li, "Software Reliability Models, " in Encyclopedia of Software Engineering, vol . 2, J.J. Marciniak, Ed.,2nd ed. New York: John Wiley & sons inc.,2002,pp. 1594-1610.

[160]. Smidts C and M.Li, "Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems, "University of Maryland, Washington D.C.NUREG/GR-0019, November 2000.

[161]. Smith, R. M. and Bain I. J. (1975) An exponential power life-testing distribution. Communications In Statistics 4(5): 469-81.

[162]. Smrithy V. , Rekha, V.Adinarayanan,Anurag Maherchandani,Sneha Aswani, "Bridging the Computer Science Skill Gap with Free and Open Source Software" International Conference on Engineering Education (ICEED 2009),Kuala Lumpur, Malaysia,2009 IEEE.

[163]. SourceForge, http://sourceforge.net.

[164]. Srinath LS. Reliability engineering. 3rd ed. New Delhi:Affiliated East West Press;1991.

[165]. Subash Bhatnagar "India's Software Industry", Technology, Adaptation, and Exports: How Some Developing Countries Got It Right, Vandana Chandra(Ed.),World Bank, 2006,Pp, 95-124.

[166]. T.R. Moss, The Reliability Data Handbook, ASME Press, 2005.

[167]. Thomas, W. C. (1973) Modeling the bathtub curve. Proceedings of the AnnualReliability and Maintainability Symposium.

[168]. Victor van Reijswound and Corrado Topi Open Source Software in Africa,2003

[169]. Vinay Tiwari and R.K. Pandey," Open Source Software and Reliability Metrics", International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 10, December 2012.

[170]. Vladimir Zelijkovic, Nela Radovanovic and Dragomir llic "Software Reliability Models and Parameter Estimation" Scientific technical review, 2011,vol.61.no.2, pp. 57-60.

[171]. Voas,J., Payne,J.: *Dependability certification of Software Components*, Journal of Systems and Software 2000, 52(2-3) pp.165-172.

[172]. W.L. Wang, Y. Wu and M.H. Chen, "An architecture-based software reliability model", Proceedings of Pacific Rim Int'l Symposium on Dependable Computing, 1999.

[173]. Wang K. S., F. S. Hsu and P. P. Liu. (2002) Modeling the bathtub shape hazard 90 rate function in terms of reliability. Reliability Engineering and System Safety 75:397-406.

[174]. Wang, K. S., E. H. Wan and W. C. Yang. (1993) A preliminary investigation of new mechanical product development based on reliability theory. Reliability Engineering and System Safety 40: 187-94.

[175]. Wang, K. S., S. T. Chang and Y. C. Shen. (1996) Dynamic reliability models for fatigue crack growth problem. Engineering Fracture Mechanics 54(4): 543-56.

[176]. www.agilemodeling.com.

[177]. www.epsma.org.

[178]. www.osalt.com

[179]. Xie, M. and C. D. Lai. (1996) Reliability analysis using an additive Weibull model with bathtub shaped failure rate function. Reliability Engineering and System Safety 52: 87-93.

[180]. Xie.M, Kim-Legng Poh, Yuan-Shun Dai, Computing systems Reliability: Models and Analysis, Hingham, MA,Usa: Kluwer Academic Publishers.pp. 71 – 113. 2004.

[181]. Y. Zhou and J. Davis, "Open source software reliability model: an empirical approach", Proceedings of the 5th Workshop on Open Source Software Engineering, May 2005, pp. 1-6.

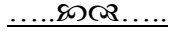
[182]. Yamada S. and S.Osaki, "Software Reliability Growth Modeling: Models and Applications," IEEE Trans. Software Eng., vol. 11, pp.1,431-1,437,1985.

[183]. Ying Zhou and Joseph Davis Open source software reliability model: an empirical approach, Fifth workshop on Open Source Software Engineering (5-WOSSE) May 2005, USA.

[184]. Yoshinobu Tamura and Shigeru Yamada, Comparison of Software Reliability Assessment Methods for Open Source Software and Reliability Assessment Tool, Journal of Computer Science vol 2 (6): pp 489-495, 2006.

[185]. Z. Jelinski and P.B. Moranda,"Software reliability research", in Statistical Computer Performance Evaluation, W. Freiberger, Ed., New York: Academic Press, 1972, pp. 465-484.

[186]. Zuzana "A brief survey of reliability growth models**",** *www.urel.feec.vutbr.cz/.../459.pdf*



Appendices

Appendix -A

DEVELOPMENT ENVIRONMENT

A.1 Introduction

The main development process is a collection of failure data and its processing for software as well as hardware. The data for software were collected from the open source sites like debian.org, bugzilla.org, and for hardware, failure data were collected from the production system of CUSAT , where Debian based server systems are used to run their website, mail server etc. Failure data were collected systematically whenever a system failure had occurred. The collected data were processed to get the required clean set of data for the study.

A.2 Bug Collection

The bug reports was collected mainly from online open source development site Debian.org. Debian is a free software consultant and offers free help through mailing lists. Debian GNU /Linux is a free operating system that comprises of 25000 packages, precompiled in a nice format. Debian GNU/Linux is developed through a distributed development all around the world. The Debian GNU/Linux distribution has a bug tracking system which consists of bugs reported by users and developers. Each bug is given a unique number as id. The bug status report initially contains the count of the total bugs, the number bugs that

have a patch, the number that are fixed and waiting to upload, the number that are being ignored, the number concerning the stable release and the number concerning the next release. Then the status is displayed. Then the actual bug report, which contains the package name, maintainer name, package id and the reported bug is displayed.

A.3 Bug Processing

Data preprocessing is an important step to refine the collected data. Generally the real world data is incomplete. Usually the collected data are incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data or may be Noisy: Containing errors or outliers or can be inconsistent: Containing discrepancies in codes or names. Data preprocessing includes data cleaning, data integration, data transformation, data reduction and data discretization.

Data cleaning usually includes fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies. where as data integration is carried out by using multiple databases, data cubes, or files. Data transformation is the normalization and aggregation process. Data reduction is reducing the volume but producing the same or similar analytical results. And Data discretization is part of data reduction by replacing numerical attributes with nominal ones. The collected bug is processed to get the required output.

The main tools used for bug processing are Web harvest (for Data Extraction), Apache (Web server), PHP and Mysql

A.4 Web Harvest

Web-Harvest is an Open Source Web Data Extraction tool *written in Java*. It offers a way to collect desired Web pages and extract useful data from them. Web-Harvest mainly focuses on HTML/XML based web sites.

Every Web site and every Web page is composed, using some logic. It is therefore needed to describe the reverse process - how to fetch desired data from the mixed content. Every extraction procedure in Web-Harvest is user-defined through XML-based *configuration files*. Each configuration file describes sequence of *processors* executing some common task in order to accomplish the final goal. Processors execute in the form of *pipeline*. Thus, the output of one processor execution is input to another one. This can be best explained using the simple configuration fragment:

```
<xpath expression="//a[@shape='rect']/@href">
  <html-to-xml>
    <http url="http://www.somesite.com/"/>
  </html-to-xml>
</xpath>
```

When Web-Harvest executes this part of configuration, the following steps occur:

1) *http* processor downloads content from the specified URL.
2) *html-to-xml* processor cleans up that HTML producing XHTML content.

3) *xpath* processor searches specific links in XHTML from previous step giving URL sequence as a result.

The bug reports extracted from Debian are initially in XML format. A Java program is developed to gather the relevant data from the XML format for further data filtering and analysis.

A.5 Bug Tracking

The Debian GNU/Linux distribution has a bug tracking system which consist of bugs reported by users and developers. Each bug is given a unique number as id.

In the bug stastus report the tags used are **P**: pending, +: patch, **H**: help, **M**: moreinfo, **R**: unreproducible, **S**: stable, **U**: upstream and **I**: lenny-ignore or squeeze-ignore. The second set of tags indicate what releases a bug applies to: O for oldstable (sarge), S for stable (lenny), T for testing (squeeze), U for unstable (sid) or E for experimental.

A.6 Conclusion

This chapter details the process of data collection and the processing of the data. The tools used are web harvest, Appache, PHP and Mysql.

Appendix -B

FREE/OPEN SOURCE PARTICIPANT SATISFACTION SURVEY

This survey has been designed to gather data to investigate factors influencing participants' satisfaction with free/open source software. It consists of 30 questions, asking about one's background, one's attitudes to free/open source software in general, and one's experience and satisfaction with one library or information management free/open source software project. It should take a person between 15 and 20 minutes to complete the survey.

Please answer all questions that apply to your current situation. If a question does not apply to you, please leave it blank, or choose 'N/A'.

In this research, free/open source software is defined as software that is issued under a license that guarantees access to source code, and ensures that users have:

1) The freedom to run the software, for any purpose;
2) The freedom to read the source code to see how it works, and to modify it to suit local conditions;
3) The freedom to redistribute copies; and
4) The freedom to improve it, and redistribute the improved version.

No data that identifies you individually is being collected, and the results of the research will not be related to specific projects, apart from indicating how many people responded to each project. The data will be used for a PhD thesis, which will be deposited in the university library and made available online in its institutional repository. The results may also be presented at conferences, or published as articles in academic or professional journals. Only aggregate data will be presented in the thesis and any publication resulting from this research, and any quote taken from comments will not be attributed.

The software used for this survey was issued under a free/open source license, in keeping with the topic of the research. The data that is provided will be stored securely in password-protected files for up to 2 years, and then it will be destroyed.

If you have questions about this survey, please contact shelbi@cusat.ac.in 09446221045

Section 1: Background and Education

1. How old are you?

◯20 or younger ◯21-25 ◯26-30 ◯31-35
◯36-40 ◯41-45 ◯46-50 ◯51-55 ◯56-60
◯61 or older

2. What is your gender?

◯Female ◯Male

3. What is your highest educational qualification?
   - None
   - Secondary or high school graduate
   - Postsecondary certificate or diploma
   - Undergraduate degree
   - Postgraduate certificate or diploma
   - Master's degree
   - PhD

4. What country do you live in?

5. How long have you been using a computer, either at work or at home?
   - < 5 years
   - 5-10 years
   - 11-15 years
   - 16-20 years
   - 21-25 years
   - 26-30 years
   - More than 30 years

6. Please rate your level of knowledge and skills in the following areas:

   Minimal | *some* | *moderate* | *much* | *extensive*

   - Knowledge and use of hardware
   - Knowledge and use of operating systems
   - Knowledge and use of one or more programming languages
   - Knowledge and use of library or information management application software
   - Ability to provide system designers with information required to develop library or information management application software
   - Ability to define library or information management application software requirements
   - Ability to assess library or information management application software features

Section 2: Attitude to Free/Open Source Software

7. Before starting this survey, how familiar were you with the idea of free/open source software?

- ○ Not at all familiar
- ○ Slightly familiar
- ○ Somewhat familiar
- ○ Quite familiar
- ○ Very familiar

8. To what extent do you use free/open source software:

not at all | very little | sometimes| often| as much as possible | don't know

- ○ On computers provided by your employer
- ○ On computers you own

9. What operating system do you use on computers provided by your employer?

10. When choosing a new application software package for use at work, to what extent do you give preference to free/open source alternatives?

- ○ It makes no difference to me
- ○ Other people make the decision for me
- ○ I prefer to use proprietary software with vendor support
- ○ I will consider a free/open source option, and choose it if it meets my needs best
- ○ I give preference to free/open source software whenever possible
- ○ I only use free/open source software
- ○ Other (please specify)

11. What operating system do you use on computers you own?

12. When choosing a new application software package for use on computers you own, to what extent do you give preference to free/open source alternatives?

- ○ It makes no difference to me
- ○ Other people make the decision for me
- ○ I prefer to use proprietary software with vendor support
- ○ I will consider a free/open source option, and choose it if it meets my needs best
- ○ I give preference to free/open source software whenever possible
- ○ I only use free/open source software
- ○ Other (please specify)

Section 3: Experience and Satisfaction with one Library or Information Management Free/Open Source Project

In this section of the survey, please answer based on one library or information management free/open source source software project you use or are involved with in some other way. Some library-related examples are DSpace, EPrints, Koha, Evergreen, Greenstone, and MyLibrary. More general information management software includes web content management software such as Drupal, wiki software such as MediaWiki or PmWiki, or blogging software such as WordPress.

If you are involved with more than one project, please choose the one that you have used or contributed to most recently.

13. What is the name of the project? There are too many to list here, so please specify the one on which you will base your subsequent responses.

14. How long have you been using or contributing to this project?

- ○ Less than 6 months
- ○ Between 6 months and one year
- ○ 1 to 2 years
- ○ 2 to 4 years
- ○ 4 to 6 years
- ○ 6 to 8 years
- ○ More than 8 years

15. How would you describe your current role in this project? Examples of roles include user, developer, maintainer, trainer, release manager, etc. If you have more than one role, please choose the one that takes up most of your time.

16. What other roles have you held in this project, if any?

17. This survey is concerned with two aspects of a free/open source software project: roles that relate to a specific implementation used in one or more institutions, and roles that relate to the wider project/developer community or the version of the software available for anyone to download. You may be involved in one or both of these aspects. Please indicate:

    None | less than 5 hours | 5-10 hours | 11-20 hours | 21-30 hours | more than 30 hours

    In the last 6 months, how many hours per week have you spent in a role relating to a specific implementation, on average?

    In the last 6 months, how many hours per week have you spent in a role relating to the wider project/developer community or the version of the software available for anyone to download, on an average?

18. What proportion of your time working on this project, either locally or on the wider project, has been part of your paid employment?

- None
- Less than 20%
- Between 20% and 50%
- Between 50% and 80%
- Between 80% and 100%

19. Which of the following activities have you carried out with this software/project?

- Please tick all that apply
- Installed the software
- Upgraded the software to a more recent release
- Studied the source code to see how it works
- Used the software
- Distributed the software to others
- Joined the project's email discussion list/forum
- Asked a question on the project's email discussion list/forum
- Answered a question on the project's email discussion list/forum
- Promoted the project by talking about it to others, for example at a conference
- Promoted the project by writing about it for publication
- Provided resources to support the project, such as hosting an email discussion list, forum, or wiki
- Organised an event relating to the project, such as a meeting or conference
- Wrote documentation to help others use the software
- Customised the software to meet local needs, either yourself, or by having a developer do so
- Reported a bug to the system developers
- Requested an enhancement from the system developers
- Contributed local changes back to the project
- Fixed one or more bugs
- Evaluated existing software functionality
- Written software to add new features

20. Have you used or contributed to this project in any other ways? Please specify.

21. Which of the following best describes how any training you have received affects your use of the software:

    n/a | *not at all* | *very little* | *somewhat* | *considerably* | *extensively*

- Training provided by outside organisations
- In-house training
- Self-study using tutorials or online help
- Self-study using manuals or other documents

22. Please briefly describe any other training you have received that affects your use of the software:

23. Please indicate your general level of satisfaction with the following characteristics of the software:

    N/A | not at all | satisfied | slightly satisfied | somewhat satisfied | quite satisfied | completely satisfied

- Reliability (i.e doesn't freeze, crash, or lose data)
- Functionality
- Free from bugs
- Easy to use
- Easy to learn
- Documentation
- Easy to install
- Easy to configure to meet local needs
- Release frequency
- Easy to add new features
- Helpfulness of community
- Security and access control

24. Please rate your experience in the following categories, relative to your perception of other people involved in the project:

    Considerably less than most | Slightly less than most | About the same as most | Slightly more than most| Significantly more than most

- Experience using this type of software
- Experience using this particular software package
- Experience using computers in general
- Experience as a member of a software development project

25. Please indicate your agreement with each of the following statements about the project's developers:

    Strongly disagree | Disagree | Neutral | Agree | Strongly agree n/a

- The project's developers are sensitive to others' needs
- The project's developers typically get right to the point when communicating with others
- The project's developers pay attention to what other people say
- The project's developers deal effectively with others
- The project's developers are easy to understand
- The project's developers generally say the right thing at the right time
- The project's developers are easy to communicate with
- The project's developers respond to messages quickly
- The project's developers express ideas clearly

26. Please indicate your agreement with the following statements about the project's culture:

    Strongly disagree | Disagree | Neutral | Agree | Strongly agree

- I feel encouraged to contribute to this project
- Anyone is encouraged to contribute to this project
- Only a few people are allowed to contribute to this project

- I find other people's contributions to this project valuable.
- Other people find my contributions to this project valuable.
- Information about the future development plans for this project is easy to find
- The future development plans for this project are clear
- The project has infrequent, formal releases of new versions of the software
- The project has frequent releases of incremental versions with bug fixes and small enhancements

27. How much influence have you had on the software features/ functionality, in your institution's local version?

- None
- Very little
- A moderate influence
- Much influence
- Very much influence

28. How much influence have you had on the software features/ functionality, in the version that is available for downloading by others?

- None
- Very little
- A moderate influence
- Much influence
- Very much influence

29. Please indicate your agreement with each of the following statements about the software:

Strongly disagree | Disagree | Neutral Agree | Strongly agree

- In comparison with other software I work with, this software has complex requirements
- This software has a complex design
- When working with this software, I have clear, planned goals and objectives for the tasks I am carrying out
- When working with this software, I know what I am responsible for
- When working with this software, I know exactly what other people expect of me

30. Are there any other comments you would like to make about your use of this software package, your involvement in the project, or reasons for your satisfaction or dissatisfaction? For example, how does it compare to other projects you are involved with?

Thank you for spending your valuable time to fill it up and send to me.

Thank you once again

Shelbi Joseph, division of Information Technology , School of Engineering, Cochin University of Science and Technology, Kochi.

.....ൠ.....

List of Publications

Papers in International journals.

[1] Shelbi Joseph, Shouri P.V. AND Jagathyraj V. P, **"**A Model for Reliability Estimation of Software based Systems by Integrating Hardware and Software", *IJCA Special Issue on "Computational Science - New Dimensions & Perspectives"* ***NCCSE, 2011.***

[2] Shelbi Joseph, Shouri P.V. AND Jagathyraj V. P, "A Simplified Model for Evaluating Software Reliability at the Developmental Stage", International Journal Software Engineering (IJSE), Volume (1): Issue (5). 2011.

Papers in National journals.

[1] Shelbi Joseph, Shouri P.V. AND Jagathyraj V. P(2010), "A Pseudo Model for Evaluating Reliability of A Computing System " , MES Journal of Technology & Management ISSN 0976-3724 Vol.01,no.02 special issue on *i*ConCEPT-2010.

Papers in International Conferences.

[2] Shelbi Joseph, G.Santhosh Kumar, Shiji S.H, Jagathy Raj V.P, "A Case Study- Role of Community and Open Source Software", International Conference on Information Technology Management at Yashwantrao Academy of Development Administration Baner Road, Pune (11th & 12th Oct. 2011).

[3] Shelbi Joseph, Shouri P.V. AND Jagathyraj V. P(2010), "A Pseudo Model for Evaluating Reliability of A Computing System" , *i*ConCEPT-2010 international conference on computational engineering practices and techniques., MES College of Engineering, Kuttipuram, Calicut, Kerala.

[4] Shelbi Joseph, G. Santhosh Kumar and Jagathyraj V.P, (2008), "A Businesscase Model for Open Source Software Development" Proceedings of the International Conference on Recent trends in computational science(ICRTCS-2008), TOC_H Institute of Science & Technology, Cochin, Kerala.

[5] Shelbi Joseph, G.Santhosh Kumar and Jagathyraj V.P,(2008), "Frequent Tree Search - A step towards XML Data Mining.", Proceedings of the International Conference on Advanced Computing & Communication Technologies for High Performance Applications, Federal Institute of Science and Technology (FISAT) Cochin, Kerala in association with IEEE & CSI.

Papers in National Conferences

[1] Shelbi Joseph, Shouri P.V. AND Jagathyraj V. P, **"**A Model for Reliability Estimation of Software based Systems by Integrating Hardware and Software", Second National Conference on Computational Science and Engineering –NCCSE.2011**.**

[2] Shelbi Joseph, G.Santhosh Kumar and Jagathyraj V.P,(2009), "A Proposed Model for Software Development Using Open Source Software", NATIONAL CONFERENCE ON FRONTIER RESEARCH AREAS IN COMPUTING (NCFRAC'09) Date: 21st FEBRUARY 2009.

[3] Shelbi Joseph, G.Santhosh Kumar and Jagathyraj V.P, P.S. Sreejith,(2008), "A study on software reliability models for Open source software development" National seminar on information, communication and intelligent systems is jointly conducted by model engineering college and IETE(institution of Electronics & Telecommunication Engineers) on 8th &9th FEB 2008.

…..ஜ۞ர…..

Curriculum Vitae

Shelbi Joseph

Chunkapurackal House,
Villa No. 13. ABC Homes,
Judgemukku, Cochin- 21,
Kerala, India.

Ph. - 9446221045
E- mail - achayanshelbi@gmail.com, shelbi@cusat.ac.in

Shelbi Joseph received the BE. Degree from University of Madras in 1992 in Computer Science and M.Tech degree in Computer Science from Department of Computer Science, National Institute of Technology, Tiruchirappalli in 2006. He spent seven years in software industry, and currently working as Assistant professor, Division of Information Technology, School of Engineering, Cochin University of Science and Technology.

He carried out his research work leading to Ph.D at School of Engineering, Cochin University of Science and Technology in Software Reliability. His areas of interest are Software Engineering, Software Reliability, Open Source Software and Data Mining. He has number of publications in National and International Journals and Conference proceedings to his credit.

.....ഇ൹.....