CHAPTER-I

Introduction

The software is becoming an integral part of the human life and the dependency is increasing with the advancement of technology, now its usages are inevitable. The use of software is inevitable to many aerospace, medical, industrial, military, and even commercial systems and the dependency over the software is increasing exponentially. Software reliability is an important factor of the software quality. A few of the reasons for increasing demand of software quality and reliability are given below (Boehm, 1991)

a) Software is the heart of many life-critical systems,
b) Software is created by humans, who can commit mistakes,
c) The software is executed machines which are error-intolerant,
d) Software development life cycle is more affected by budget and time rather than reliability.

When software has become integral part of human life and the dependency is further increasing day by day, in such situation, quality of the software system is foremost necessary.

1.1 Software Quality

For successful businesses that develop software, quality cannot be an optional, it must be the basic requirement. "The Quality is the degree to which a system, component, or process meets customer or user needs or expectations" (IEEE Standard 729, 1983). Quality is an important factor of a system at the present time and the survival and sustaining of the organization in the fast changing era is largely depends on the quality of its products.

To ensure and manage quality, it is required to *plan*, determine the standards to be followed; *perform*, implement the plan to ensure that they are achieved; and *monitor*, verify results to confirm that the standards have been meet and identify and remove problems that affect the quality of the system.

The software quality is measured in terms of functionality, usability, testability, adaptability, maintainability and reliability. All these quality attributes are independent of each others. All the attributes are important from the quality aspect. The reliability attribute of a software system is one of the most important and demanding feature of the quality.

1.2 Software Reliability

Reliability shows the trustworthiness or dependability of the software product. Reliability is the probability with which the software system work "correctly" for the given period of time. It is true that software with more defects is unreliable. It is also true that the reliability improves by removing the detected defects and defects get reduced over a period of time.

Software reliability is the probability that the given software will perform the intended task without causing failure for the given time interval under specified environment. Here, failure is the incapability of performing a task as specified in the requirement document. Since reliability of software is a probabilistic statement, it is important to define the term and condition under which the statement about the reliability is made. (Xie, 1991).

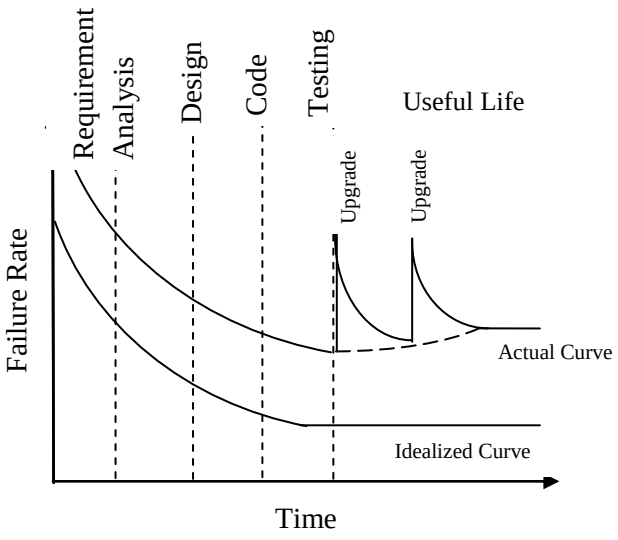

Figure 1.1 Software Life Cycle Curve

Software is not susceptible to the age and work environment whereas the hardware product is susceptible to the age and environmental factors. Hence, we can conclude that the software life cycle curve is free from age and environmental effects. Hence, the failure rate curve for should follow the "idealized curve", as depicted in figure 1.1. The undetected faults results in high failure rates in the initial phase of life of a program. Once these faults are attended, the curve flattens. Due to the upgrades in the software product, the failure rate increases drastically. The upgrade may be due to any of the reasons like additional features, technology changes or inherent faults. The failure rate decreases gradually, partly when the defects are identified and corrected. (Pressman, 2001).

1.3 Application of Reliability

The reliability of a project can be measured for different purposes. The estimation and prediction are two frequently and interchangeably used terms, so the main difference is that; Reliability estimation tells about the current value, whereas the prediction attribute forecasts the value of the reliability at a future stage or point of time.

In addition to the estimation and prediction, reliability measurements can be used for c*ertification* means to i.e. system acceptability. It helps management in optimal allocation of resources and the criteria when to stop the testing and so that the product can be delivered to the customer for the field use. It also provides a degree of confidence to the customer about the product going to use. It is therefore concluded that the reliability estimations and predictions are of very helpful to control the software processes.

1.4 Why Software Reliability measurement is difficult?

The software reliability measurement is difficult due to the following reasons:

a) The amount of improvement of reliability on fixing a single error depends on the utilisation of code where the error is located in the software.
b) The measured reliability depends on the observer.
c) Due to the fault removal process the reliability of a product keeps changing.

The reliability estimation as well as prediction for the software system is quite a difficult task. Until now, we still do not have ideal and perfect methodology for measuring software reliability. Software reliability measurement is still a difficult task because of two reasons, first understanding of the nature of software is difficult task and second, the nature and structure of the software is diversified.

Software reliability is a probabilistic measure and assessing reliability of during testing is one of the important aspects of reliable software development process. A large number of reliability models exits and are being used at present. A reliability growth model is a technique to assess the reliability quantitatively and plays an important role. For assessing software reliability quantitatively and development of highly reliable software product, software reliability models play an important role.

It is important to elaborate that the software are produced by human beings, who is prone to commit mistakes, therefore the delivered products still have errors and are imperfect. Here imperfect means there is difference between the results expected by the software system to the results produced. These discrepancies are known as *faults*. The faults are attributed to the system due to ignorance or miss interpretation of user requirements, rules of the computing

environments, requirements not communicated properly between developer and the actual user. The incapability of software to perform the desired operation and provide the specified performance as required is known as *failure* (IEEE Standard 729, 1983).

1.5 Reliability Metrics

The requirements of reliability may be different for different software products. The requirements of a software product must be mentioned clearly in the SRS including for the reliability. To measure the reliability quantitatively, metrics are used. The basic metrics used for the reliability measurement are discussed below:

a) Mean Time To Failure (MTTF) measures the average time between two successive failures, the failure data need to be recorded for large failures. Mathematically, MTTF can be given as

$$MTTF = \sum_{i=1}^{n} \frac{t_{i+1} - t_i}{(n-1)} \qquad 1\text{-}1$$

b) Mean Time To Repair (MTTR) is the time being taken to fix an error. It takes the average of the time elapsed in diagnosis and fixing the errors which is responsible for failure.

c) Mean Time Between Failures (MTBR) is the sum of MTTF and MTTR.

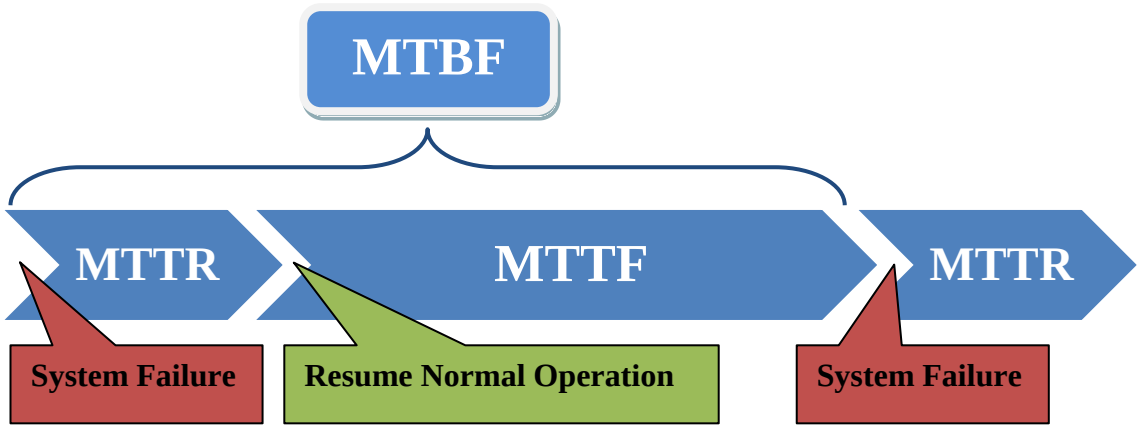

Figure 1.2 A relation between MTBF, MTTD and MTTR

d) Rate of occurrence of failure (ROCOF) provides the frequency of failure noticed during the execution/ operation.

e) Probability of Failure on Demand (POFOD) tells when a particular system will fail at the time of requirement; PFFOD provides the likelihood of the system.

f) Availability is the probability that the system is available and produces the output when demand*ed*, mathematically,

$$\text{Availability} = \frac{\text{System up time}}{\text{System up time + System down time}}$$
$$= \frac{\text{MTTF}}{\text{MTTF+MTTR}}$$

g) Maintainability is the ease with which the failed system can be repaired and brought to the original or specified conditions in a given period of time by following the specified procedures and resources.

1.6 Software Reliability Models

The probability is the chances of the system to fail to perform the intended task. Software is free from age and environmental impacts and hence should not wear out and continue to perform even after undesired output. More than 200 models have been developed since last four decades, but how to measure reliability quantatively is still a difficult and cumbersome task. There is no single model exists which suits every software products, in every situation. Many software models contain, assumptions, factors which affect the assumptions and mathematical function.

1.6.1 Reliability Growth Models

A Software Reliability Growth Model (SRGM) is an approach, which tells how reliability of a software system can be improved as errors are detected and corrected, mathematically. SRGMs help to predict and measure the reliability. Hence, SRGM guide to determine when to stop testing, on reaching a particular reliability level. Researchers have developed a many SRGMs in the past 35 years to measure reliability. The reliability measurement includes the number of remaining faults, failure rate, and reliability of the software. Selecting the best SRGM for the given software is a difficult task for researchers. Number of tools and techniques are available in the literature, which help to select the optimum SRGM. Due to the limited number of model selection criteria, the existing tools and techniques cannot be used with high confidence (Sharma, Garg, & Nagpal, 2010).

1.6.2 Steps of Reliability Growth Modelling

In the software testing, programs are executed with the desired and undesired inputs and the outputs obtained from the execution is monitored for the correctness.. The incorrect output is counted as a failure. Faults which are responsible for the failure are identified and eliminated. The troubleshooting process during the software testing is called fault-removal process. During the testing, reliability of the software gets increased fault is removed. The process of reliability improvement while the testing is in progress is known as *reliability growth* (Xie, Software Reliability Modelling, 1991). It is therefore the reliability estimation or prediction is a sequential process and having the following steps.

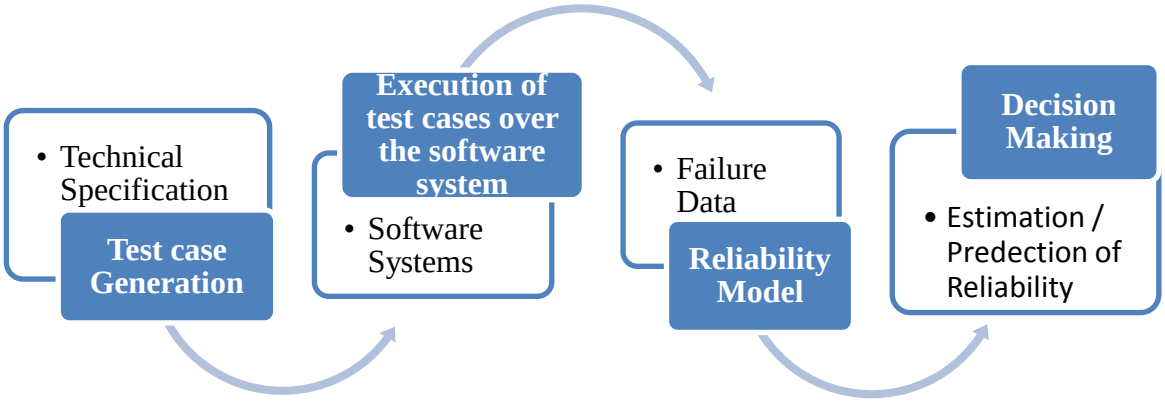

Figure 1.3 Software Reliability Measurement

As shown in the figure 1. 3, software reliability measurement follows a series of activities. The software reliability measurement consists of four major steps:

a) Technical specification or software requirement specification contains detail information about the purpose of the product, software system and the test cases are generated from the specification.
b) The test cases are needed to be executed to the software system under test.
c) Verify the output of the test cases and failure data is collected. Based on the requirements of reliability model, failure data is collected i.e the occurrences and time of each fault.

The collected failure data is applied to the concerned reliability model, which produces the estimation of the reliability.

1.7 Parameter Estimation Techniques

Parameter estimation is a technique to evaluate the value of the unknown variables by applying on specified model on the collected dataset. The parameter values are selected based on how perfectly it fits the observed data. There are two widely used techniques of parameter estimation (Myung, 2002).

1.7.1 Least-squares estimation (LSE)

This technique of parameter estimation is applied to various statistical concepts such as linear regression, sum of squares error, proportion variance, and root mean squared deviation. It doesn't require distributional assumptions like MLE, for obtaining a descriptive measure for the purpose of summarizing observed data.

1.7.2 Maximum likelihood estimation (MLE)

It is a standard statistical technique used for parameter estimation. MLE have various attractive attributes, it have good convergence properties (nearly always) as the training samples grows. It is very simple than other existing methods like Bayesian techniques etc.

1.8 Research Objective

The proposed research has been carried out with following objectives:

a) Literature review carried out, may help in future study and analysis for the researchers who want to keen to contribute to the software reliability.
b) In order to classify and compare the existing software reliability models, the time between failures and failure counts, is further classified on the behalf of prefect and imperfect debugging.
c) A new variant of existing Jelinski-Moranda reliability growth model, considers a new methodology for the imperfect debugging.
d) The time-domain approach also called failure count method provides better accuracy for the parameter estimation with the present tools, but extra efforts are required for collecting such failure data compare to the interval-domain approach. An intermediate way between the cost of data collection and the reliability accuracy must be found by the model developers.

1.9 Outlines

SECOND CHAPTER reviews the literature of the classification of models and explains the requirement of classification and provided a brief about the categories of the reliability models to help the customer as well as testers to take the decision, which category, his/ her model falls. The comparison criteria are also mentioned in this chapter which defines which model is to be chosen in the selected category.

THIRD CHAPTER explains the research work carried out in the field of software reliability. The existing models have been classifyed in the two widely used categories, i.e. failure rate models and failure counts which is based on failure history of the software and testing datasets/ results. Models are further classified based on the occurrences of failure with respect to the time and code. A lot of models have been proposed by reliability researchers with the perfect debugging and imperfect debugging, so these models have been classified in these two sub categories.

FOURTH CHAPETER explains the proposed model which elaborates the assumptions followed in designing a new model for the prediction/ estimation of the reliability. The mathematical derivation and parameter estimation is also explained along with the brief on the tool used for model simulation.

FIFTH CHAPTER will depict the results obtained from the model/ methodology proposed in previous chapter. The comparison with other models with the given datasets will tells about the usefulness and the effectiveness of the proposed model..

SIXTH CHAPTER explains the conclusion and future scope of research and followed by references.

CHAPTER-II

Literature Review on Software Reliability

2.0 Introduction

This chapter presents an overview on the requirement of software reliability classification and classification categories based on the testing strategy, assumptions, failure data, software structure etc. and reliability models published recently. This chapter also explains the parameter estimation techniques which are used in calculation of software reliability.

2.1 Requirement of Classification

Software reliability models are being used for a long time and till today various models have been proposed, discussed, modified and generalized by the reliability researchers whereas the some models have been criticized. The existing software reliability classification has been reviewed briefly. Need of classification scheme:

a) Software reliability models classification is useful in comparison of different reliability models.

b) Different sets of models make it easy to obtain new models which are more practical compare to the present models by identifying the hypothetical assumptions, which are far from reality, made for these existing models.

c) Help managers/management to select a group of software reliability models based on their requirement. This classification empowers managers to reject more than 80% of software reliability models as per the requirement and choose the particular model out of 20% remaining model.

2.2 Classification of Software Reliability Models

The area of SRGM is not too old compare to the conventional engineering discipline and even to the hardware reliability. Enough research has already been carried out in the area of software reliability. Various researchers have already been proposed different models to measure the reliability, but there is no model exists which suits all the software systems. There are various methods exists to ensure the suitability of the model. To ease the model selection process, the available models have been studied by various researchers by classifying them in different categories. The few categories of existing models are given below:

2.2.1 An interesting classification proposed by the author (Musa & Okumoto, 1984) to divide the existing models into different sections depending on the number of failures that can observed is a finite time or not. The models which consider the finite number of faults in the assumptions are falls under finite failure category, whereas, number of faults are not fixed and are infinite are falls under the infinite failure category. There may exist a software system which have infinite number of faults, comes under infinite failure category.

2.2.2 The author Goel and Botsani (1985) defined four main categories based on the failure history of the models. In this classification, nature of the failure also kept in to consideration. The author considers the four main categories, which includes failure rate, failure count, fault seeding and input domain based models.

2.2.3 Ramamoorthy et al (Ramamoorthy & Bastani, 1986) has divided SRGMs into two categories based on the failure data. There are types of failure data, discussed in the next section, fault-counting and non-fault-counting. The fault-counting model category includes number of remaining faults and the fault occurrence rate.

2.2.4 Musa and Okumoto et al, (Musa, Iannino, & Okumoto, 1987) have developed a classification scheme, which allows a relationship within the same group to be established for models and shows where model development has occurred. They have classified models in term of five different attributes, which includes type, Time domain, category, class and family.

2.2.5 A new classification is used by the author Mellor (Mellor, 1987b), where existing models have been used as a family tree of black box models and structure models. The category of structure models is subdivided into inter-failure time and fault manifestation models. However, author defines that most of the statistic models are exponential order in nature, which is a kind of unification for various models.

2.2.6 M Xie (Xie, 1991) has classified models in to various categories, some of them had already considered by other authors. The categories considered by the author are Markov,

Non-Homogeneous Poisson process, Bayesian, Statistical Data Analysis Methods, Input-domain Based, Seeding and tagging models based on probabilistic assumptions.

2.2.7 The Author (Wood, 1996) has grouped software reliability models based on the shape of the graph of the defect detection rate. The author has classified models into concave and S-shaped Models. The thing to be noticed here is that the both models have the same asymptotic behaviour, i.e., it is important to note here that the rate of defect detection decreases gradually as the fault are removed.

2.2.8 KS Trivedi (Trivedi, 2001) classifies existing software reliability models in to two categories. The first category is Data-domain models, which focuses on the fault contents of the software product. It is sub-divided into error seeding models and input domain based models. The Time-domain models is second category considered by the author., The failure history of the software and time required to detect the errors are kept in to consideration, in this category. These models are further sub-divided as Homogeneous Markov, Non-homogeneous Markov, Semi-Markov and others.

2.2.9 The author (Pham, System Software Reliability, 2006) has classified the software reliability models in to two main categories the deterministic and the probabilistic. The Probabilistic model emphasised on the failure occurrences and the fault removals probability, whereas the deterministic model consider the number of distinct operators and operands, errors, machine instructions in the program etc.

2.2.10 Software development life cycle based classification has been carried out by the author (Sharma & Garg, 2011), where one category can expand to one or more phases of SDLC. The author has classified the models in each phase of SDLC.
The early prediction models considers the characteristics from requirement phase to testing and these observed characteristics are extrapolated to which predict the software behaviour during operation. The SRGMS are similar to the early prediction, except that it does not considers the failure behaviour of software in all the phases instead it considers only testing phase, and extrapolates the behaviour during operation, which is subdivided in to failure rate models, and NHPP models. The other categories considered by Sharma at. el. includes Input

Domain, architecture, hybrid black box and white box Based Models. The interesting thing here is that the author has combined more than one phase in to the single category.

2.2.11 The author (Razeef & Nazir, 2012) has considered four classes of reliability models which includes design phase models, Unit Testing phase models, Integration testing phase models, and Acceptance testing & Operational Phase models. A reliability model can be applied on more than one phase of software development life cycle.

It is concluded from the above literature that the existing models based on different assumptions, finite or infinite failures, the type of datasets used for estimation of reliability, models based on the SDLC phases, behavior of the failure, software structure, graph of failure rates etc. However, there is no standard classification which is accepted globally. Hence the following table provides the summary of the classifications proposed so far.

Table 2.1 Classification of Software Reliability Models

S.No	Name of Author(s)	Year	Classifications
1.	JD Musa, K Okumoto	1984	Finite Failures Infinite Failures
2.	AL Goel and Botsani	1985	Failure Rate Models, Failure Count Models, Fault Seeding Models Input Domain Based Models
3.	CV Ramamoorthy, FB Bastani	1986	Fault-Counting Models Non-Fault-Counting
4.	JD Musa, A Iannino, & K Okumoto	1987	Time Domain, Category, Type, Class And Family
5.	P Mellor	1987	Black Box Model Structure Model Inter-Failure Time Model Fault Manifestation Models
6.	M Xie	1991	Markov Models, Non-Homogeneous Poisson Process Models, Bayesian Models, Statistical Data Analysis Methods, Input-Domain Based Models, Seeding And Tagging Models
7.	KS Trivedi	2001	Data-Domain Models

			Time-Domain Models Homogeneous Markov, Non-Homogeneous Markov, Semi-Markov And Others
8.	H Pham	2006	Deterministic Models Probabilistic Models
9.	Alan Wood	1996	Concave And S-Shaped
10.	Kapil Sharma, RK Garg	2010	Early Prediction Models Software Reliability Growth Models Input Domain Based Models Architecture Based Models Hybrid Black Box Models Hybrid White Box Models
11.	Razeef Mohammad and Mohsin Nazir	2012	Design Phase Models, Unit Testing Phase Models, Integration Testing Phase Models, Acceptance Testing and Operational Phase Models

2.3 Model Selection Criteria

The goodness of the model can be measured based on its ability to predict the future behaviour of the software from the available failure data. The effectiveness of SRGMs can be ensured comparing the proposed model based on the following criteria:

The common approach is to use all models and measure the reliability from each, and then choose the one which gives better results, but this approach is very expensive and time consuming, hence doesn't suits the purpose. The criterion to choose the best model is is difficult to decide. It is still difficult to decide which model to be selected, like model with best or worst results, or the model which provides the result according to the management's requirement to keep all happy or the couple of models give the same result. The criteria's used to compare the reliability models are being described as follows:

2.3.1 The Bias

It is the sum of the difference between the estimated curve and the actual data.

$$Bias = \frac{\sum_{i=1}^{k} (\hat{m}(t_i) - m_i)}{k} \qquad 2\text{-}1$$

2.3.2 The Mean Square Error (MSE)

The mean square error (MSE) measures the deviation between the predicted values with the actual observations, and is defined as

$$MSE = \frac{\sum_{i=1}^{k} (m_i - \hat{m}(t_i))^2}{k - p} \qquad 2\text{-}2$$

2.3.3 The Mean Absolute Error (MAE)

The mean absolute error (MAE) is similar to MSE, but the way of measuring the deviation is by the use of absolute values. It is defined as

$$MAE = \frac{\sum_{i=1}^{k} \left| m_i - \hat{m}(t_i) \right|}{k - p} \qquad 2\text{-}3$$

2.3.4 The Mean Error of Prediction (MEOP)

The mean error of prediction (MEOP) sums the absolute value of the deviation between the actual data and the estimated curve, and is defined as

$$MEOP = \frac{\sum_{i=1}^{k} \left| \hat{m}(t_i) - m_i \right|}{k - p + 1} \qquad 2\text{-}4$$

2.3.5 The Accuracy of Estimation (AE)

The accuracy of estimation (AE) can reflect the difference between the estimated numbers of all errors with the actual number of all detected errors. It is defined as

$$AE = \left| \frac{M_a - a}{M_a} \right| \qquad 2\text{-}5$$

where, M_a and *'a'* are the actual, and estimated cumulative number of detected errors after the test, respectively.

2.3.6 The Noise

The noise is defined as

$$Noise = \sum_{i=1}^{k} \left| \frac{\lambda(t_i) - \lambda(t_{i-1})}{\lambda(t_{i-1})} \right| \qquad 2\text{-}6$$

2.3.7 The Predictive-Ratio Risk (PRR)

The predictive-ratio risk (PRR) which measures the distance of model estimates from the actual data against the model estimate, is defined as

$$PRR = \sum_{i=1}^{k} \frac{\hat{m}(t_i) - m_i}{\hat{m}(t_i)} \qquad 2\text{-}7$$

2.3.8 The variance

The variance which is the standard deviation of the prediction bias, is defined as

$$Variance = \sqrt{\frac{1}{k-1} \sum_{i=1}^{k} (m_i - \hat{m}(t_i) - Bias)^2} \qquad 2\text{-}8$$

2.3.9 The Root Mean Square Prediction Error (RMSPE)

The Root Mean Square Prediction Error (RMSPE) is a measure of the closeness with which the model predicts the observation. It is defined as

$$RMSPE = \sqrt{Variance^2 + Bias^2} \qquad 2\text{-}9$$

2.3.10 RSQ

The Rsq can measure how successful the fit is in explaining the variation of the data. It is defined as

$$Rsq = 1 - \frac{\sum_{i=1}^{k} (m_i - \hat{m}(t_i))^2}{\sum_{i=1}^{k} (m_i - \sum_{i=1}^{k} m_j / n)^2} \qquad 2\text{-}10$$

2.3.11 The Sum of Squared Errors (SSE)

The sum of squared errors (SSE) is defined as

$$SSE = \sum_{i=1}^{k} (m_i - \hat{m}(t_i))^2 \qquad 2\text{-}11$$

2.3.12 The Theil Statistic (TS)

The Theil statistic (TS) is the average deviation percentage over all periods with regard to the actual values. The closer Theil's Statistic is to zero, the better the prediction capability of the model. It is defined as

$$TS = \sqrt{\frac{\sum_{i=1}^{k} (\hat{m}(t_i) - m_i)^2}{\sum_{i=1}^{k} m_i^2}} X100\% \qquad 2\text{-}12$$

2.3.13 Distance Based Approach

The development of the DBA method begins with defining the optimal state of the overall objective, and specifies the ideally good values of attributes involved in the process. The OPTIMAL is simply the SRGM that has all the best values of attributes. The objective function for finding such a solution can be formulated as

$Minimise\ \delta\{Alt(x), OPTIMAL\}$

$Subject\ to\ x \varsigma X$

2.4 Failure Datasets

The failure data is collected based on time and interval of the failure. The time of fault is considered in the first called time-domain approach whereas the second approach emphasised on failures in the given time-interval. These data is analysed by practitioners for predicting reliability in the applications. Some models works with both types of datasets. The failure datasets are considered by the reliability practitioner and researchers to predict the reliability and analyse the failure behaviour, which gives the feedback on the software system for which the failure data was collected.

2.4.1 Time Domain Dataset

The time-domain approach emphasis and considers the times at which failure has been noticed and records it.

Table 2.2 Time-domain Dataset

Failure Records	Time Between Failure (Min.)	Actual Failure Time (Cumulative) (Min.)
1	25	25
2	30	55
3	15	70
4	15	95
5	17	112
6	7	119
7	26	143
8	44	187

2.4.2 Interval-Domain Dataset

The interval-domain approach counts the number of failures noticed in a continuous time interval like test session, minutes, hour, weeks, and days. The approach provides the more than one failure in the given time interval.

Table 2.3 Interval-domain Dataset

Time (Hours)	Observed Number of Failures
1	2
2	4
3	1
4	1

The author (Pham, 2006) has emphasised that the time-domain approach provides more accurate parameter estimation results compare to the interval-domain data, but the data recording involves extra efforts. Therefore the researchers should balance the cost of data collection with the accuracy of reliability required. Availability of real-time dataset with accuracy is a difficult task and Most of the companies/ institutes and organisations afraid of sharing actual datasets of their software due various reasons.

CHAPTER-III

Software Reliability Growth Models

3.0 Introduction

Assurance on reliability of the software is highly desirable for the satisfaction of the user, but without knowing the initial error it is very difficult to predict the quality of the software. Enough research has been done in the past few years to ensure the level of reliability of the software and various tools and techniques have already been developed and established to estimate and predict the errors, failure rate, MTTF and the reliability of the software. The SRGMs also provides the feedback of the residual number of defects exists in the software and the reliability increases on gradually when the detected errors are corrected.

Most of the models have the number of parameters as the initial parameter. For the finite number of faults focus is mainly given to the remaining number of faults. Efforts have been made to estimate the reliability at the early phase of the software development but this getting the analytical method foe such purpose is very much difficult for the moderate size software, because the numbers of faults are not the measure of software reliability that we can measure. For estimating the current failure intensity, a number of models have been already developed but in general it is not an easy task. Failure intensity also plays a vital role for the prediction of faults apart from the number of faults.

Most of the models have considered the debug capability of the software in their assumption during the proposal of the model, which gives a fair idea to classify the existing models in to two categories, perfect debugging and imperfect debugging.

a) Perfect Debugging:

In perfect debugging, it is assumed that the fault is removed perfectly i.e. no residual of fault left back, as and when it is detected/ noticed and new fault are not allowed to introduce, as the effect of fault removal process, is known as perfect debugging.

b) Imperfect Debugging Models:

The assumptions of Jelinski-Moranda model which says that the faults are removed immediately as and when it is detected without introducing new errors seems to be impractical. When detected errors are removed, there may be the chances to introduce new errors. The probability of finding errors is high initially, but gets reduced subsequently on the

removal of the errors, hence we can say, it is proportional to the number of remaining errors in the system. There are two possibilities of introduction of new errors while removing:

i. It may be possible that the error is not removed perfectly and the probability is that the some percentage of the error still remains. Suppose p is the probability of the removal of the fault then, 1-p is the probability of the fault remains in the program.

ii. The other possibility is that the error removal process might have introduced some new errors. These errors are independent of the above fault removal process. It may get introduced in both the cases, where fault is removed perfectly and fault is not removed perfectly as well.

3.1 A New Classification of Reliability Models

The software reliability growth models divided in to two main categories based on dataset being used by the model i.e. the time- between failure models and fault count models. These models are further classified as perfect debugging and imperfect debugging models, based on assumption of fault removal process.

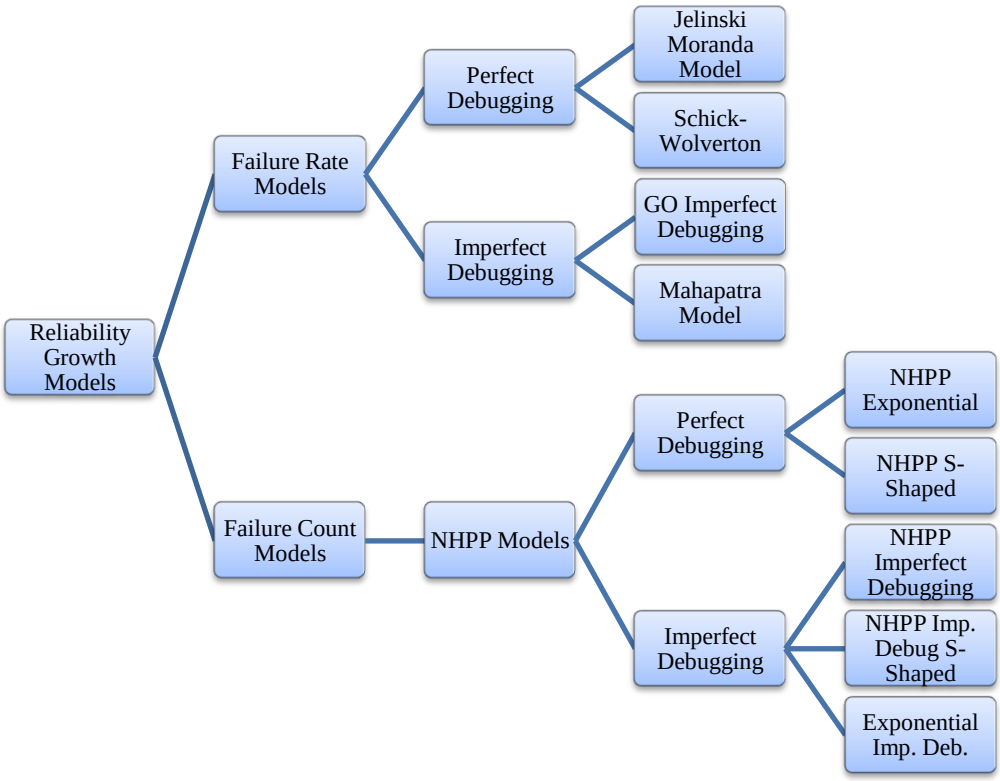

Figure 3.1 Classification Categories of SRGMs

I. Failure Rate Models Or Time between Failure Models

In failure rate models, the time between two consecutive failures say, $(i-1)^{th}$ and the i^{th} failures follows a distribution. The distribution parameters depend on the number of faults remaining in the program during the interval and obtained from the values of times between failures in the given intervals and this is done by using parameter estimation techniques like MLE, LSE etc.

A. Perfect Debugging Models

1) Jelinski-Moranda Model
2) Schick-Wolverton Model
3) Modified Schick-Wolverton Model
4) Lipow Modified Jelinski –Moranda Model
5) Joe and Reid Model
6) Decreasing Failure Intensity Model
7) ShanthiKumar General Markov Model
8) Variable Fault Exposure Coefficient Model
9) Jelinski-Moranda Geometric Model
10) Moranda Geometric Poisson Model
11) Littlewood-Verrall Baysian Model
12) A Quantum Modification to the JM Model
13) Variable Fault Exposure Coefficient Model

B. Imperfect Debugging Models

1) Goel-Okumoto Imperfect Debugging Model
2) Mahapatra's Modified Jelinski-Moranda Model

II. Failure Count Models

It is clear from the name of the category itself; this class of models considers the number of faults occurred in the given time-interval. We can estimate the parameters from the observed values of failure counts or it can also be estimated from failure times The failure count models are further classified in categories based on the assumption of debugging capabilityof the model.

A. Perfect Debugging:

1. NHPP Exponential Models

1) Goel-Okumoto Model
2) Simple Generalised Goel-Okumoto Model
3) Musa Exponential Growth Model
4) Musa Logarithmic Poisson Execution Time Model
5) Hyper Exponential Growth Model
6) Yamada-Osaki Exponential Growth Model
7) Yamada Exponential Model
8) Yamada Rayleigh Model

2. NHPP S-Shaped Models

1) Inflection S-shaped Model
2) Delayed S-shaped Model
3) Connective NHPP Model

B. Imperfect Debugging Models

1. NHPP Imperfect Debugging Models

1) Yamada Imperfect Debugging Model

2. *NHPP Imperfect Debugging S-shaped models*

1) Pham-Nordman-Zhang Model
2) Pham Nodrman Generalised NHPP Model

3. Exponential Imperfect Debugging Model

1) Pham-Exponential Imperfect Debugging Model

3.2 Failure Rate Models OR Time between Failure Models

The failure rate models, also called time between failure models, have been divided in to two categories, perfect and imperfect debugging.

The Perfect Debugging Models

The failure rate models are explained below in details:

3.2.1 Jelinski-Moranda Model

The Jelinski–Moranda model (Jelinski & Moranda, 1972), which is cited many times, for describing the failure behaviour of a software system, is one of the widely used and oldest models. The model considers the process of discovery and the removal of faults in computer software and was introduced in 1972.

The assumptions in this model include the following:

a) The program contains "N" initial faults which is an unknown but fixed constant.

b) Each fault in the program is independent and equally likely to cause a failure during a test and has same impact of failure.

c) Time intervals between occurrences of failure are independent of each other.

d) Whenever a failure occurs, a corresponding fault is removed with certainty and no new faults are inserted during the removal of the detected fault.

e) The software failure rate during a failure interval is constant and is proportional to the number of faults remaining in the program.

The program failure rate at the i^{th} failure interval is given by

$$\lambda(t_i)=\Phi[N-(i-1)], \text{ Where } i = 1,2,.....N \qquad 3\text{-}1$$

Where, N = the number of initial faults in the program, Φ = a proportional constant, the contribution any one fault makes to the overall program, t_i = the time between the $(i-1)^{th}$ and the i^{th} failures.

$$\lambda(t_i) = \phi\,[N-(i-1)], \quad \text{Where } i = 1,2,.....N$$

The Probability Density function is given as

$$f(t_i)=\lambda(t_i)e^{-\int_0^{t_i}\lambda(x_i)dx_i}$$

$$=\phi[N-(i-1)]e^{-\int_0^{t_i}\phi\,[N-(i-1)]dx_i}$$

$$f(t_i)=\phi[N-(i-1)]e^{-\phi\,[N-(i-1)]t_i} \qquad 3\text{-}2$$

The Cumulative Density function is given as

$$f(t_i)=\int_0^{t_i} f(x_i)dx_i$$

$$=\int_0^{t_i}\phi[N-(i-1)]\, e^{-\phi[N-(i-1)]x_i}dx_i$$

$$f(t_i)=1-e^{-\phi[N-(i-1)]t_i} \qquad 3\text{-}3$$

Then the reliability is given by

$$R(t_i)=1-f(t_i)$$

$$R(t_i)=e^{-\phi[N-(i-1)]t_i} \qquad 3\text{-}4$$

Now we have to estimate the value of N and assume that the value of ϕ is known.

Let t1, t2, ..., tn is a failure data set

$$L(N) = \prod_{i=1}^{n} f(t_i)$$

$$= \prod_{i=1}^{n} \phi[N-(i-1)]e^{-\phi[N-(i-1)]t_i}$$

$$L(N) = \phi^n \prod_{i=1}^{n} [N-(i-1)]e^{-\phi[N-(i-1)]t_i}$$

On taking Natural Log on both side

$$\ln(L(N))=\ln[\phi^n \prod_{i=1}^{n} [N-(i-1)]e^{-\phi[N-(i-1)]t_i}]$$

$$=n \ln \phi + \sum_{i=1}^{n} \ln[N-(i-1)] - \phi \sum_{i=1}^{n} [N-(i-1)]t_i$$

Taking first derivates on w.r.t N

$$\frac{\delta \ln(L(N))}{\delta N} = 0 + \sum_{i=1}^{n} 1/[N-(i-1)] - \phi \sum_{i=1}^{n} [t_i]$$

Set the value $\frac{\delta \ln(L(N))}{\delta N} = 0$

$$\sum_{i=1}^{n} 1/[N-(i-1)] = \phi \sum_{i=1}^{n} [t_i]$$

In case, where the value of N and ϕ both are unknown.

$$L(N,\phi) = \phi^n \prod_{i=1}^{n} [N-(i-1)]e^{-\phi \sum_{i=1}^{n} [N-(i-1)]t_i}$$

On taking the natural Log on both sides

$$\ln(L(N,\phi))=\ln[\phi^n \prod_{i=1}^{n} [N-(i-1)]e^{-\phi \sum_{i=1}^{n} [N-(i-1)]t_i}]$$

$$=n \ln \phi + \sum_{i=1}^{n} \ln[N-(i-1)] - \phi \sum_{i=1}^{n} [N-(i-1)]t_i$$

Taking first partial derivative w.r.t. N and ϕ, first w.r.t. N

$$\frac{\delta \ln(L(N,\phi))}{\delta N} = 0 + \sum_{i=1}^{n} 1/[N-(i-1)] - \phi \sum_{i=1}^{n} [t_i]$$

Set the value $\frac{\delta \ln(L(N,\phi))}{\delta N} = 0$

$$\sum_{i=1}^{n} 1/[N-(i-1)] = \phi \sum_{i=1}^{n} [t_i]$$

$$\phi = \frac{\sum_{i=1}^{n} 1/[N-(i-1)]}{\sum_{i=1}^{n} [t_i]} \qquad 3\text{-}5$$

Now taking derivative w.r.t. ϕ

$$\frac{\delta \ln(L(N,\phi))}{\delta \phi} = \frac{n}{\phi} + 0 - \sum_{i=1}^{n} [N-(i-1)t_i]$$

Set the value $\frac{\delta \ln(L(N,\phi))}{\delta \phi} = 0$

$$\frac{n}{\phi} = \sum_{i=1}^{n} [N-(i-1)t_i]$$

$$\phi = \frac{n}{\sum_{i=1}^{n} [N-(i-1)]t_i} \qquad 3\text{-}6$$

From equation (4-5) and (4-6)

$$\sum_{i=1}^{n} \frac{1}{[N-(i-1)]} = \frac{n \sum_{i=1}^{n} [t_i]}{\sum_{i=1}^{n} [N-(i-1)]t_i}$$

$$\sum_{i=1}^{n} \frac{1}{[N-(i-1)]} = \frac{n}{N - \left(\frac{1}{\sum_{i=1}^{n} [t_i]}\right) \sum_{i=1}^{n} (i-1)t_i} \qquad 3\text{-}7$$

3.2.1.2 Schick-Wolverton Model

The Schick-Wolverton (Schick & Wolverton, 1978) model is a modification of the basic Jelinski-Moranda model. The assumptions are similar to the Jelinski-Moranda model except that, the failure rate at the i^{th} time interval raises with time t_i. The failure rate between the $(i-1)^{th}$ and the i^{th} failure can be articulated as,

$$\lambda(ti)=\Phi[N-(i-1)]ti, \text{ Where } i = 1,2,.....N \quad 3\text{-}8$$

Where, Φ and N are the same as in the Jelinski-Moranda model and t_i is the test time since the $(i\text{-}1)^{th}$ failure. The PDF can be given as

$$f(t_i)=\Phi[N-(i-1)]\, t_i\, e^{-\frac{\Phi[N-(i-1)]\, ti^2}{2}} \quad 3\text{-}9$$

The CDF can be given as

$$F(t_i)=1-e^{-\frac{\Phi[N-(i-1)]\, ti^2}{2}} \quad 3\text{-}10$$

And the Reliability can be defined as

$$R(t_i)=e^{-\frac{\Phi[N-(i-1)]\, ti^2}{2}} \quad 3\text{-}11$$

3.2.1.3 Modified Schick-Wolverton Model

This model has emerges by modifying the Schick-Wolverton model (Sukert, 1977), who considered more than one failure at each time interval. The failure rate function is expressed as

$$\lambda(ti) = \Phi[N-n_{i-1}]ti, \text{ Where } i = 1,2,.....N \quad 3\text{-}12$$

where $n_{(i-1)}$ *is* the cumulative number of failures at the $(i\text{-}1)^{th}$ failure interval. The software reliability function can be given as

$$R(t_1) = e^{-\Phi[N-n_{i-1}]\frac{ti^2}{2}} \quad 3\text{-}13$$

3.2.1.4 Lipow Modified Jelinski–Moranda Model

The author has changed the Jelinski-Moranda and assumes that the errors may not be corrected immediately when discovered. This modification was made for real time development project.

$$\lambda(t_i) = 1/\{k[N - EC(p)]\} \quad 3\text{-}14$$

and the reliability is given by

$$R(t_i) = e^{-\{k[N-EC(p)]\}} \quad 3\text{-}15$$

Where, N is number of inherent errors, k is constant of proportionality and EC(p) is the corrected errors till the p^{th} testing period.

3.2.1.5 Joe and Reid Model

The author (H. Joe, 1985) has suggested an alternative way for the Jelinski-Moranda and Littlewood software reliability models. The author has considered failure times instead of inter-failure times.

3.2.1.6 Decreasing Failure Intensity (DFI) Model

The general Markov model, proposed by Xie and Bergman (Xie & Bergman, 1988) have considered the fault detection probability which depends on the fault size. The authors considered that faults are of different size, it practical, it is not true. The author has assumed that the earlier faults are having a high detection probability.

The failure intensity λ (i) is a function of the number of remaining faults, that is

$$\lambda(i) = \phi[N_0-(i-1)]^{\alpha}, \quad \text{for } i=1,2,.....,N_0 \qquad 3\text{-}16$$

Where, λ(i) is defined as the rate of the occurrence of the next failure after the removal of the $(i\text{-}l)^{th}$ fault. λ(i) should decrease fast at the beginning and the decrease becomes slower for each i.

3.2.1.7 Shanthikumar General Markov Model

The Jelinski-Moranda model can be generalized by using a general time-dependent transition probability function (ShanthiKumar, 1981). Denoted by N(t) the number of faults that are detected and removed during time [0,t). Suppose that there are No initial software faults and the model assumes that, after n faults are removed, the failure intensity of the software is given by

$$\lambda(n,t) = \phi(t)(N_0-n) \qquad 3\text{-}17$$

Where, ϕ(t) is a proportionality factor. Note that this model reduces to the Jelinski-Moranda model, if ϕ(t) is a constant independent of t.

3.2.1.8 Variable Fault Exposure Coefficient Model

An interesting model with variable FEC which has the same advantages as the DFI model discussed above is presented by the author (Bittanti, 1988b). It assumes that a large number of trivial faults are detected earlier and the last faults are hard to detect which implies that the decrease of the failure intensity becomes less. It contradicts the Jelinski-Moranda's assumption that all faults are equally exposed in testing by a more realistic one.

The variable FEC model assumes that λ_j is the number of remaining faults multiplied by k(j), a function of j, that is

$$\lambda_j = k(j)(N_0 - j) \qquad 3\text{-}18$$

3.2.1.9 Jelinski-Moranda Geometric Model

The Jelinski-Moranda Geometric model (Moranda 1979) assumes that the program failure rate function is initially a constant *D* and decreases geometrically at failure times. The program failure rate and reliability function of time-between-failures at the ith failure interval can be expressed, respectively, as

$$\lambda(t_i)=Dk^{i-1} \qquad 3\text{-}19$$

Where, *D* is initial program failure rate and *k is* parameter of geometric function ($0 < k < 1$)
If we allow multiple error removal in a time interval, then the failure rate function becomes

$$\lambda(t_i)=Dk^{n_i-i} \qquad 3\text{-}20$$

Where, n_{i-1} is the cumulative number of errors found up to the $(i\text{-}1)^{th}$ time interval.

3.2.1.10 Moranda Geometric Poisson Model

The Moranda geometric Poisson model (Moranda 1975) assumes fixed times *T*, 2*T*, of equal length intervals, and that the number of failures occurring at interval *i*, *Ni*, follows a Poisson distribution with intensity rate Dk^{i-1}.
The probability of getting m failures at the ith interval is

$$\Pr\{N_i = m\} = \frac{e^{-Dk^{i-1}}(Dki-1)m}{m!} \qquad 3\text{-}21$$

The reliability and other performance measures can be easily derived in the same manner as in the Jelinski-Moranda model.

3.2.1.11 Littlewood-Verrall Baysian Model

It uses different approach to the development of a model (Littlewood & Verral, 1979) for times between failures. This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors. It also models the fact that as errors are repaired, the average improvement in reliability per repair decreases. It treat's an error's contribution to reliability improvement to be an independent random

variable having Gamma distribution. This distribution models the fact that error corrections with large contributions to reliability growth are removed first. This represents diminishing return as test continues.

It is considered that the software reliability should not be specified in terms of number of errors in the program, instead the time between failures are considered and are assumed to follow an exponential distribution. The parameter of this distribution, i.e. failure rate is treated as a random variable following gamma distribution,

$$f(t_i / k_i) = k_i e^{-k_i t_i} \qquad 3\text{-}22$$

4.2.1.12 Negative-binomial Poisson Model

Assume that the intensity λ is a random variable with the gamma density function having parameters *k* and *m*, that is,

$$f(\lambda)=\frac{1}{\Gamma(m)} k^m \lambda^{m-1} e^{-k\lambda} \qquad 3\text{-}23$$

3.2.1.13 Littlewood's Stochastic Reliability-Growth Model

An assumption (Littlewood, 1981) that the failure rate of a program is a constant multiple of the (unknown) number of faults remaining. This implies that all faults contribute the same amount to the failure rate of the program. The assumption is challenged and an alternative proposed. The suggested model results in earlier fault-fixes having a greater effect than later ones (the faults which make the greatest contribution to the overall failure rate tend to show themselves earlier, and so are fixed earlier), and the DFR property between fault fixes (assurance about programs increases during periods of failure-free operation, as well as at fault fixes).

3.2.1.14 A Bayesian modification to the Jelinski-Moranda SRGM

The author (Littlewood & Sofer, 1987) uses Bayesian method for estimating the parameters instead of MLE and shows that it is sometimes an improvement on Jelinski-Moranda.
However, both versions have a tendency to give optimistic answers, probably owing to a key, but implausible, underlying assumption common to both models. The authors conclude that the generally poor performance of the models is such that they should only be used with great caution**.**

3.2.1.15 A Quantum Modification to the Jelinski-Moranda Model

A modified Jelinski-Moranda model (Ho, Chan, & Chung, 1991) is proposed, which assumes that different types of faults may have different effects (measured by the size of failure-quantum) on the failure rate of the software. Therefore, fault removals make the failure rate decrease differently for different faults. The size of the failure-quantum of a fault can be determined by the structure of the software. For example, if a fault is located in a very frequently executed module, then its removal should make the failure rate of the software decrease faster (a larger size of failure-quantum). The failure intensity function is given by

$$\lambda_i = (Q - \sum_{j=1}^{i-1} w_j)\psi \qquad 3\text{-}24$$

Where, Q = an unknown constant representing the initial number of failure-quantum units inherent in a software, Ψ = the failure rate corresponding to a single failure-quantum unit, and W_i = the number of failure-quantum units of the i^{th} fault.

3.2.1.16 Optimal Software Release Based on Markovian SRM

The optimal software release problems (Rinsaka & Dohi, 2004) based on Jelinski-Moranda SRM has been explained, and revisited the optimal software release policies by taking account of a waste of software testing time. Authors have formulated the total expected software costs with two different release policies, and compare them in terms of the cost minimization. It can be concluded that the existing optimal software release policy underestimates and overestimates the real optimal software release time and its associated cost function, respectively.

Then the total expected software cost $V_1(T)$ is formulated as

$$V_1(T) = c_1 a[1 - e^{-\phi T}] + c_2 a e^{-\phi T} + c_3 T \qquad 3\text{-}25$$

3.2.2 Imperfect Debugging Models

These models consider that the fault removing process is not perfect. The Jelinski-Moranda model was developed assuming the debugging process to be perfect which implies that there is one-to-one correspondence between the number of failures observed and faults removed. But in reality, it is possible that the fault which is supposed to have been removed may cause a new failure.

3.2.2.1 Goel-Okumoto Imperfect Debugging Model

Goel and Okumoto (Goel & Okumoto, 1979b) extend the Jelinski-Moranda model by assuming that a fault is removed with probability *p* whenever a failure occurs. The failure rate function of the J-M model with imperfect debugging at the i^{th} failure interval becomes

$$\lambda(ti) = \Phi[N-p(i-1)], \quad \text{Where } i = 1,2,.....N \qquad 3\text{-}26$$

Where ϕ and N are similar to Jelinski-Moranda Model and p is the probability of correction of error.

3.2.2.3 Modified Jelinski-Moranda Model with Imperfect Debugging

The assumptions of modified Jelinski-Moranda Model (G. S. Mahapatra, 2012) are similar to the Jelinski-Moranda model except that it does not consider the perfect debugging process in fault removal activity. The modified J- M model assumes that the debugging process is truly imperfect. It assumes that whenever a failure occurs, the detected fault is removed with probability p, the detected fault is not perfectly removed with probability q, and the new fault is generated with probability r. So it is obvious that $p + q + r = 1$ and $q \geq r$. The software failure rate function between the $(i-1)^{th}$ and i^{th} failure for our modified Jelinski-Moranda model with imperfect debugging is given by

$$\lambda(t_i) = \phi[N - p(i-1) + r(i-1)] = \phi[N - (i-1)(p-r) \qquad 3\text{-}27$$

Where, ϕ, N and t_i have the same meaning as defined in the Jelinski-Moranda model.

The reliability function at the failure interval is given by

$$R(t_i) = 1 - F(t_i) = e^{-\phi[N-(i-1)(p-r)t_i} \qquad 3\text{-}28$$

Note that if p=1 and r=0, then the failure behavior of the modified model becomes the same as the Jelinski-Moranda model. Thus, the Jelinski-Moranda model may be regarded as a special case of this modified model.

3.2.3 Summary of Evolution of Failure Rate Growth Models

It is learnt from the above literature review that the Jelinski – Moranda Model introduced in 1972 was the first one in the field of reliability growth models history and all other models have been emerged from Jelinski-Moranda Models.

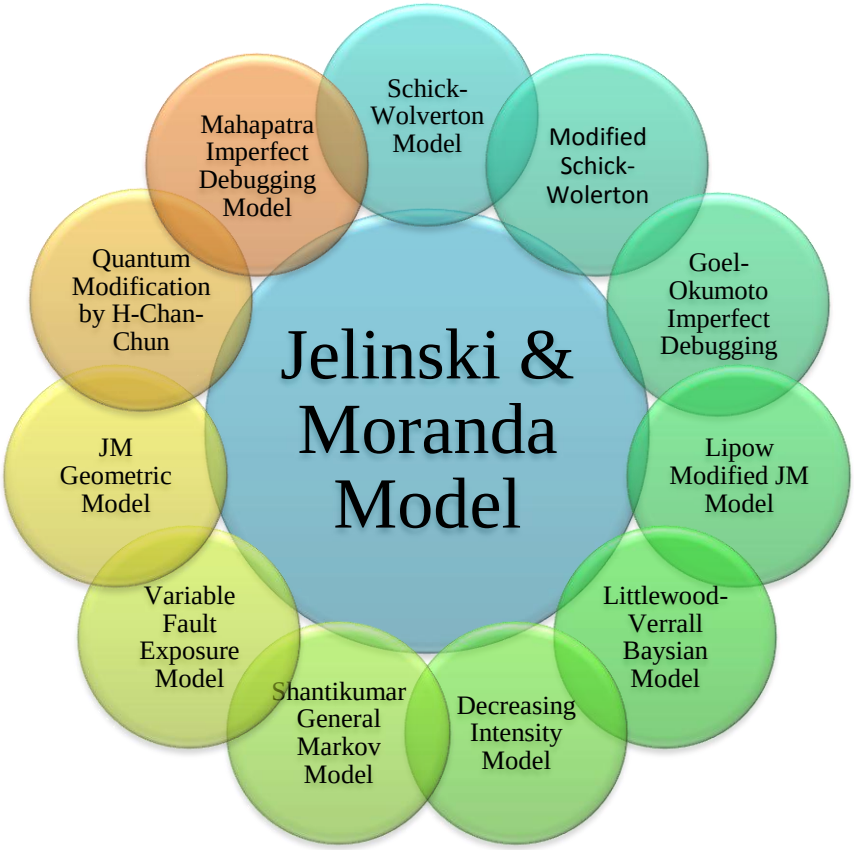

Figure 3.2 Evolutions of Failure Rate Growth Models

The time between failure models or failure rate models uses the dataset of the time-domain approach, which involves recording the individual times at which failure occurred, as illustrated in Table 2.2. A summary of existing failure rate software reliability models is given below in table 3.1.

Table 3.1 Summary of failure rate software reliability models

SN	Model Name	Author(s) Name	Year	Failure Rate Function	Assumptions
1.	The Jelinski-Moranda (J-M) model	Z. Jelinski, P. B. Moranda	1972	$\lambda(t_i)=\Phi[N-(i-1)]$	1. The No. of initial software faults is an unknown, but a fixed constant. 2. A detected fault is removed immediately and no new fault is introduced. 3. Times between failures are independent exponentially distributed random quantities. 4. All remaining s/w

					faults contribute the same amount to the s/w failure intensity. 5. The s/w failure rate during a failure interval is constant and proportional to the no. of faults remaining in the program.
2.	Schick-Wolverton Model	GJ Schick, RW Wolverton	1978	$\lambda(t_i)=\Phi[N-(i-1)]t_i$	1. It is similar to the J-M model. 2. It assumes that the failure rate at the i^{th} time interval increases with time t_i since the last debugging
3.	Modified Schick-Wolverton Model	AN Sukert	1977	$\lambda(t_i) = \Phi[N-n_{i-1}]t_i$	1. It is a modification to the Schick-Wolverton Model. 2. It allow more than one failure at each time interval
4.	Goel-Okumoto Imperfect Debugging Model	AL Goel, K Okumoto	1979	$\lambda(t_i) = \Phi[N-p(i-1)]$	1. It extend the J-M model 2. a fault is removed with probability *p* whenever a failure occurs
5.	Lipow Modified Jelinski -Moranda Model	Lipow		$\lambda(t_i) = 1/\{k[N - EC(p)]\}$	1. Errors are not corrected immediately
6.	Littlewood-Verrall Baysian Model	B Littlewood, JL Verrall	1979	$f(t_i / k_i) = k_i e^{-k_i t_i}$	1. This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors. 2. Reliability should not be specified in terms of number of errors in the program, instead the times between failures are considered. 3. It is assumed to follow an exponential distribution.
7.	Shanthikumar General Markov Model	JG Shanthikumar	1981	$\lambda(n,t) = \phi(t)(N_0-n)$	A Generalization of JM Model
8.	Decreasing Failure Intensity (DFI) Model	M Xie, B Bergman	1988	$\lambda(i) = \phi[N_0-(i-1)]^{\alpha}$	fault-detection probability depends on the size of fault earlier failures are caused by faults having a high detection

					probability
9.	A Variable Fault Exposure Coefficient Model	S Bittanti	1988	$\lambda_j = k(j)(N_0 - j)$	It assumes that a large number of trivial faults are detected earlier and the last faults are hard to detect It contradicts the JM's assumption.
10.	Jelinski-Moranda Geometric Model	Z. Jelinski, P. B. Moranda	1979	$\lambda(t_i) = Dk^{i-1}$	the program failure rate function is initially a constant *D* and decreases geometrically at failure times
11.	Moranda Geometric Poisson Model	P. B. Moranda	1975	$\Pr\{N_i = m\} = \frac{e^{-Dk^{i-1}}(Dk_{i-1})m}{m!}$	It assumes fixed times *T*, 2*T*, of equal length intervals, and that The number of failures occurring at interval *i*, *Ni*, follows a Poisson distribution with intensity rate Dk^{i-1}.
12.	Negative-binomial Poisson Model			$f(\lambda) = \frac{1}{\Gamma(m)} k^m \lambda^{m-1} e^{-k\lambda}$	Assume that the intensity λ is a random variable with the gamma density function
13.	Quantum Modification to the JM Model	Tsu-Fens Ho, Wah-Chun Chan, and Chyan-Goei Chung	1991	$\lambda_i = (Q - \sum_{j=1}^{i-1} w_j)\psi$	It assumes that different types of faults may have different effects
14.	Optimal Software Release Based on Markovian Software Reliability Model	K. Rinsaka, T. Dohi	2004	$V_1(T) = c_1 a[1 - e^{-\phi T}] + c_2 a e^{-\phi T} + c_3 T$	It considers optimal software release policies by taking account of a waste of software testing time.
15.	Modified JM Model with Imperfect Debugging Phenomenon	GS Mahapatra	2012	$\lambda(t_i) = \phi[N - p(i-1) + r(i-$ $= \phi[N - (i-1)(p - r)$	It assumes that whenever a failure occurs, the detected fault is removed with probability p, the detected fault is not perfectly removed with probability q, and the new fault is generated with probability r

3.3 Failure Count Models

As it has been explained in earlier sections, models which fall in this category counts number of faults or failures in specified time intervals instead of times between failures. The failure counts follow a time-dependent discrete or continuous failure rate which is known as stochastic process.

Non-homogeneous Poisson Process Models

A stochastic NHPP is a Poisson process with failure rate parameter $\lambda(t)$ is a function of time t. NHPP model is required to calculate an appropriate mean value function to denote the projected number of failures practiced up to a certain time. Probabilities of a given number of failures for the NHPP model are calculated by a generalization formula: Where, N (t) is the number of events by time t, m (t) is the mean for failure data, k = 0,1,2,3,... (Pham, System Software Reliability, 2006).

The Non-Homogeneous Poisson Process (NHPP) group of models provides an analytical framework for describing the software failure phenomenon during testing. The main issue in the NHPP model is to estimate the mean value function of the cumulative number of failures experienced up to a certain point in time.

The NHPP models are further classified based on assumption of perfect debugging and imperfect debugging. Goel-Okumoto Model is considered as the basic NHPP Model, which is further generalized and assumptions are analyzed to be more realistic in the present software development and testing environment.

3.3.1 NHPP Perfect Debugging Model

These models considers that the fault removal process is perfect and instant i.e. fault is removed perfectly 100% and not new faults are introduced as a side effect of the removed fault.

3.3.1.1 Goel-Okumoto Model

Goel & Okumoto (Goel & Okumoto, 1979) has described the time dependent finite exponential class of model, which is one of the first NHPP model proposed. This model is based on Poisson distribution assumes that faults are independent and have the same chance

of being detected and each time a failure occurs, the error which caused it is immediately removed, and no new errors are introduced in the software.

The mean value function solution of the differential equation is given by

$$m(t) = a(1-e^{-bt}), \text{ Where } a>0, b>0 \qquad 3\text{-}29$$

Where, *a* & *b* are parameters to be determined using collected failure data. The intensity function λ(t) defined as the derivative of m(t) is then

$$\lambda(t) = \frac{dm(t)}{dt} = abe^{-bt} \qquad 3\text{-}30$$

3.3.1.2 Generalization of Goel-Okumoto Model

It has been observed that the software failure intensity increases slightly at the beginning and then it begins to decrease. The only difference between the Goel's generalized NHPP model (Goel, 1985) and the GO-model is that the mean value function of the generalized NHPP model is given by

$$m(t) = a[1-e^{-bt^c}] \qquad 3\text{-}31$$

Where, *'a'* is the expected number of faults in the software, *'b'* is a kind of scale parameter which reflects the intensity of testing and *'c'* is another parameter which can be interpreted as the test quality. The failure intensity function of this model is

$$\lambda(t) = \frac{dm(t)}{dt} = abct^{c-1}e^{-bt^c} \qquad 3\text{-}32$$

This generalized model is a three-parameter model and with an appropriate value of the third parameter, it can give better goodness-of-fit than the original model.

3.3.1.3 Musa Execution Time Model

Musa and Okumoto (Musa, 1987) Model is a binomial type model. This model is referred to as the exponential model and presented a theory of software reliability analysis based on the execution time instead of calendar time. The execution time is a better measure of time.

Mathematically,

$$\lambda(\tau) = fK[N_0 - \mu(\tau)] \qquad 3\text{-}33$$

Where f and K are parameters related to the testing phase and No and $\mu($ t) the number of initial faults in the software and the number of faults corrected after t amount of testing measured in execution time, respectively.

3.3.1.4 Musa Logarithmic Poisson Execution Time Model

The author has proposed a logarithmic time model (Musa & Okumoto, 1984) for reliability measurement considering the possibility of infinite number of faults. The failure intensity decreases exponentially as a function of the number of removed faults.

3.3.1.5 Hyper Exponential Growth Model

The author (Ohba, 1984), assume that a program has a number of clusters of modules with different initial number of errors and a different failure rate.

The mean value function of the hyper exponential class NHPP model is

$$m(t) = \sum_{i=1}^{n} a_i[1 - e^{-b_i t}] \qquad 3\text{-}36$$

Where, n = number of clusters of modules, ai = number of initial faults in cluster i, bi = failure rate of each fault in cluster i. The failure intensity function is given by

$$\lambda(t) = \sum_{i=1}^{n} a_i b_i e^{-b_i t} \qquad 3\text{-}37$$

3.3.1.6 Yamada-Osaki Exponential Growth Model

The author, Yamada – Osaki (Yamada & Osaki, 1985), proposed that SRGMs are classified in terms of software reliability growth index of the error detection rate per error. Yamada et al. discussed existing SRGMs including Exponential SRGM, Modified exponential SRGM, delayed s-shaped, Inflection s-shaped, logistic, Gompertz growth curve models.The expected number of faults detected for the entire software can be obtained as

$$m(t) = a\sum_{i=1}^{k} p_i[1 - e^{-b_i t}] \qquad 3\text{-}38$$

NHPP S-Shaped Models

In the NHPP S-shaped model, the growth curve is an S-shaped curve which means that the curve crosses the exponential curve from below and the crossing occurs once and only once. It assumes that the error detection rate differs among faults and fault is removed immediately when it is detected without introducing new errors.

3.3.1.7 Delayed S-shaped Model

The Authers (Yamada, Ohba, & Osaki, 1983) have proposed an s-shaped stochastic model for a software error detection process, where the growth curve of the number of detected software errors or the mean value function for the observed data is S-shaped. And the error detection model is NHPP.

3.3.1.8 Connective NHPP Model

Nakagawa (Nakagawa, 1994) proposed a model, called the connective NHPP model, where the basic shape of the growth curve is exponential and that an S-curve forms due to the test. In the connective NHPP model, a group of modules called "main route modules" are tested first, followed by the rest of the modules. Even if the failure intensity of the faults in the main route module and the other modules are similar, the growth curve becomes an S-curve since the search for their detection starts at different points in time.

3.3.2 Imperfect Debugging Models

NHPP Imperfect Debugging Models

Software reliability models with imperfect debugging processes and a constant error detection rate $b(t)=b$, compare to the perfect debugging models with $a(t) =a$, studied by Yamada (Yamada 1984). The NHPP imperfect debugging model is based on the basic assumptions that say, it is possible to introduce new errors, when detected errors are removed, and the probability of finding an error in a program is proportional to the number of remaining errors in the program.

3.3.2.1 Yamada Imperfect Debugging Model

Yamada assume that the new errors may be introduced during fault removal process and the probability of finding an error in a program is proportional to the number of remaining errors in the program.

NHPP Imperfect Debugging S-shaped models

3.3.2.2 A Generalized Imperfect-debugging Fault-detection Model

The generalized NHPP imperfect-debugging fault-detection rate model was formulated by Pham et al. (Pham & Normann, 1997b). The author considers that the error detection rate varies among different faults and the fault is immediately removed on detection and new faults can be introduced.

3.3.2.3 Pham-Nordmann-Zhang Model

Pham et al in (Pham, Nordmann, & Zhang, 1999b) assumes that the function of fault introduction rate is a linear time-dependent, whereas the fault detection rate is non-decreasing time-dependent with an inflection S-shaped model.

The mean value function is given by

$$m(t) = \frac{a}{1+\beta e^{-bt}}\left([1-e^{-bt}]\left[1-\frac{\alpha}{\beta}\right]+\alpha t \right) \qquad 3\text{-}45$$

Exponential Imperfect Debugging Model

3.3.2.4 Pham Exponential Imperfect Debugging Model

Pham (Pham, 2000a) assumes that the fault introduction rate is an exponential function of testing time, whereas detection rate function is non-decreasing with an inflection S-shaped model.

The mean value function is given by

$$m(t) = \frac{\alpha b}{b+\beta}\left(\frac{e^{(\beta+b)}-1}{e^{bt}+c} \right) \qquad 3\text{-}46$$

A lot of research is in progress in the field of failure count especially in Non-Homogeneous Poisson Process models. The new models are being developed and implemented on various datasets to estimate and predict the reliability of the given system.

CHAPTER-VI

Proposed Reliability Growth Model

4.0 Fault Dependent Imperfect Debugging Model

The Jelinski–Moranda model (Jelinski & Moranda, 1972), which is cited many times, for describing the failure behaviour of a software system, is one of the widely used and oldest models also known as black-box models. The model considers the process of discovery and the removal of faults in computer software and was introduced in 1972. It is important to emphasise that the Jelinski-Moranda model considers only positive things, hence the model always provides over optimistic results of reliability prediction. It has been enumerated in the literature that the main reason for the shortcoming of this model is its assumption i.e. the failure rate is constant, fault is removed perfectly, faults are independent from each other and each fault has the same impact of the failure etc.

The Jelinski-Moranda model was developed with the consideration that the debugging process is perfect which implies that faults are removed immediately as and when it is detected. It may be possible that the fault which is supposed to have been removed may not be removed perfectly. In the proposed modified Jelinski-Moranda model, we consider that whenever a failure occurs, the cause of the failure identified and the detected fault is removed immediately but not perfectly without delay. However, the correction of one fault may have the impact over the other faults too, as the faults are dependent on each other. In addition to this, the failure removal process provides the better understanding to the programmer and he may correct other remaining errors in part or full, in addition to the current error.

Software troubleshooting is the process of scanning, identifying, diagnosing and resolving faults, errors and bugs in software. It is a systematic process that aims to filter out and resolve problems, and restore the software to normal operation. The code of the program is checked thoroughly, which gives a better understanding to the programmer. As the programmers understanding about the logic and semantic of the program increases, he can identify other problems, also (if exists). In addition to this, faults are dependent on each other, which mean the effect of correction of one error may correct other error too.

The proposed model has over comes the drawbacks of Jelinski-Moranda Model, single fault is attended/ corrected at a time and the faults are removed perfectly, of basic Jelinski–Moranda Model, which is not true in practical. Hence, the newly proposed model considers the multiple failure correction at a time with imperfect debugging.

4.1 Assumptions

The devised model assumes that:

- The program contains "N" initial faults which is an unknown but a fixed constant.
- Faults in the program are dependent and do not have same impact of failure.
- Whenever a failure occurs, a corresponding fault is removed immediately with probability p, without adding new error(s).
- The rectification of one fault can affect the other faults too i.e. other faults or the part of the fault may also get rectified with some probability.
- The software failure rate during a failure interval is constant and is proportional to the number of faults remaining in the program.
- The failure rate at the i^{th} time interval decreases with time t_i since the last debugging.

The total faults exists in the program is constant and the program contains N initial faults, but the value of the constant, N is unknown in the advance. It is assumed here that the faults are dependent on each other, which means the correction of one fault may improve the other fault in full or in part. The failure removal process is not perfect that means there are the chances that the part of the fault is still left back, in other words fault is removed with the probability p where $0 \leq p \leq 1$. Each fault is assumed to have different impact over the failure, thus the failure rate vary from fault to fault but for a single failure is assumed to be a constant.

It is learnt from the literature (Goel & Okumoto, 1979b) and (G. S. Mahapatra, 2012) that whenever a failure occurs, the cause of the failure is identified and removed immediately with probability p and the p has been kept constant. Keeping p constant seems to be impractical, as each fault will have the different impact and faults are dependent, which means correction of one fault will have the impact on other, but it will not be same always. It is therefore, the mean of the probabilities of all the remaining faults are taken in to consideration, in order to measure the probability of i^{th} fault, being removed at the time t_i. The probability of fault could not be removed is q=1-p. It is assumed that the new errors are not

introduced during the troubleshooting process. The failure rate between the $(i-1)^{th}$ and i^{th} failure for our modified Jelinski-Moranda model with fault dependent imperfect debugging is given by

$$\lambda(t_i) = \phi\,[N-(\frac{\sum_{j=i}^{n} p_j}{n-i})(i-1)], \qquad \text{Where } i = 1,2,.....N \qquad 4\text{-}1$$

Where, ϕ is the a constant of proportionality, N is the number of faults available initially in the software, p is the probability of fault removal process and t_i is the time-interval of $(i-1)^{th}$ and i^{th} failure.

4.2 Derivation

The failure rate function is given by

$$\lambda(t_i) = \phi\,[N-(\frac{\sum_{j=i}^{n} p_j}{n-i})(i-1)], \qquad \text{Where } i = 1,2,.....N$$

The Probability Density function is given as

$$f(t_i)=\lambda(t_i)e^{-\int_0^{t_i}\lambda(x_i)dx_i}$$

$$=\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]e^{-\int_0^{t_i}\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]dx_i}$$

$$f(t_i)=\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]e^{-\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i} \qquad 4\text{-}2$$

The Cumulative Density Function is expressed as

$$F(t_i)=\int_0^{t_i} f(x_i)dx_i$$

$$=\int_0^{t_i}\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]e^{-\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]x_i}dx_i$$

$$F(t_i)=1-e^{-\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i} \qquad 4\text{-}3$$

Then the reliability is given by

$$R(t_i)=1-F(t_i)$$

$$R(t_i)=e^{-\phi\,[N-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i} \qquad 4\text{-}4$$

4.3 Parameter Estimation

The parameters are estimated by sung the MLE method for the value of *N* and ϕ in the modified Jelinski-Moranda model.

Suppose t_1, t_2 ,t_3 ....., t_n is given failure dataset as in the Jelinski-Moranda model. The likelihood function of N and ϕ is given by

$$L(N,\phi) = \prod_{i=1}^{n} f(t_i)$$

$$= \prod_{i=1}^{n} \hat{\phi}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]e^{-\hat{\phi}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i}$$

$$L(N,\phi) = \hat{\phi}^n \prod_{i=1}^{n}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]e^{-\hat{\phi}\sum_{i=1}^{n}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i}$$

On taking the natural Logarithm on both sides

$$\ln(L(N,\phi))=\ln[\hat{\phi}^n \prod_{i=1}^{n}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]e^{-\hat{\phi}\sum_{i=1}^{n}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i}]$$

$$=n \ln \hat{\phi} +\sum_{i=1}^{n}\ln[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]-\hat{\phi}\sum_{i=1}^{n}[\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i$$

By taking the first partial derivative with respect to N and ϕ respectively,

$$\frac{\delta \ln(L(N,\phi))}{\delta N} = 0+\sum_{i=1}^{n} 1 \Big/ [\hat{N}-\frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)] -\hat{\phi}\sum_{i=1}^{n}[t_i]$$

Set the value $\frac{\delta \ln(L(N,\phi))}{\delta N} = 0$

$$\sum_{i=1}^{n} \frac{1}{[\hat{N}-(\frac{\sum_{j=i}^{n} p_j}{n-i})(i-1)]} = \hat{\phi}\sum_{i=1}^{n}[t_i]$$

$$\hat{\phi} = \frac{\sum_{i=1}^{n} \frac{1}{[\hat{N} - (\frac{\sum_{j=i}^{n} p_j}{n-i})(i-1)]}}{\sum_{i=1}^{n} [t_i]} \qquad 4\text{-}5$$

Now taking derivative w.r.t. ϕ

$$\frac{\delta \ln(L(N,\phi))}{\delta \phi} = \frac{n}{\hat{\phi}} + 0 - \sum_{i=1}^{n} [\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i$$

Set the value $\frac{\delta \ln(L(N,\phi))}{\delta \phi} = 0$

$$\frac{n}{\hat{\phi}} = \sum_{i=1}^{n} [\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i$$

$$\hat{\phi} = \frac{n}{\sum_{i=1}^{n} [\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i} \qquad 4\text{-}6$$

From equation (5-5) and (5-6)

$$\sum_{i=1}^{n} [\frac{1}{\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)}] = \frac{n \sum_{i=1}^{n} [t_i]}{\sum_{i=1}^{n} [\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i}$$

The value of the $\hat{N}$ can be estimated from the following equation, which is independent of $\hat{\phi}$.

$$\sum_{i=1}^{n} [\frac{1}{\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)}] = \frac{n}{[\hat{N} - (\frac{1}{\sum_{i=1}^{n} [t_i]}) \sum_{i=1}^{n} \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)t_i]}$$

The above equation can be further generalized as

$$\frac{1}{N_0} + \frac{1}{N_0 - 1} + \frac{1}{N_0 - 1} + ... + \frac{1}{N_0 - n + 1} = \frac{n \sum_{i=1}^{n} [t_i]}{\sum_{i=1}^{n} [\hat{N} - \frac{\sum_{j=i}^{n} p_j}{n-i}(i-1)]t_i} \qquad 4\text{-}7$$

By putting the value of $\hat{N}$ in to equation (5-6), we can get the value of $\hat{\phi}$.

The software reliability function can be obtained as follows

$$R(t_{i+1})=1-F_{n+1}(t_{i+1})=e^{-\phi\,[N-(p*n)]t_{i+1}} \qquad 4\text{-}8$$

The estimated mean time to failure for the $(n+1)^{th}$ fault is

$$\hat{MTTF}=\frac{1}{\hat{\lambda}}=\frac{1}{\hat{\phi}[\hat{N}-\sum_{j=i}^{n} p_j n]} \qquad 4\text{-}9$$

4.4 Tools

The failure data of the software is analyzed with the help of MATLAB. The MATrix LABoratory is based on vectors and matrices. MATLAB is widely used tool for mathematical simulation and modeling. The details can be found online from http://www.mathworks.com.

CHAPTER-V

Results and Discussion

5.1 Failure Dataset

The proposed model, a new variant of Jelinski-Moranda Model, is applied to the Real-time Control System's failure data. The failure dataset contains total 136 entries of time-between failures (in sec.) given in Table 5.1 taken from (Pham, System Software Reliability, 2006), Page 157. The dataset is a time between failures, which provides the better estimation/ prediction of the failure rate and reliability compare to the failure count dataset.

Table 5.1 Real-Time Control System Data

Fault	TBF	Cum.TBF	Fault	TBF	Cum.TBF	Fault	TBF	Cum.TBF
1	3	3	**46**	193	7837	**91**	724	30085
2	30	33	**47**	6	7843	**92**	2323	32408
3	113	146	**48**	79	7922	**93**	2930	35338
4	81	227	**49**	816	8738	**94**	1461	36799
5	115	342	**50**	1351	10089	**95**	843	37642
6	9	351	**51**	148	10237	**96**	12	37654
7	2	353	**52**	21	10258	**97**	261	37915
8	91	444	**53**	233	10491	**98**	1800	39715
9	112	556	**54**	134	10625	**99**	865	40580
10	15	571	**55**	357	10982	**100**	1435	42015
11	138	709	**56**	193	11175	**101**	30	42045
12	50	759	**57**	236	11411	**102**	143	42188
13	77	836	**58**	31	11442	**103**	108	42296
14	24	860	**59**	369	11811	**104**	0	42296
15	108	968	**60**	748	12559	**105**	3110	45406
16	88	1056	**61**	0	12559	**106**	1247	46653
17	670	1726	**62**	232	12791	**107**	943	47596
18	120	1846	**63**	330	13121	**108**	700	48296
19	26	1872	**64**	365	13486	**109**	875	49171
20	114	1986	**65**	1222	14708	**110**	245	49416
21	325	2311	**66**	543	15251	**111**	729	50145
22	55	2366	**67**	10	15261	**112**	1897	52042
23	242	2608	**68**	16	15277	**113**	447	52489
24	68	2676	**69**	529	15806	**114**	386	52875
25	422	3098	**70**	379	16185	**115**	446	53321
26	180	3278	**71**	44	16229	**116**	122	53443
27	10	3288	**72**	129	16358	**117**	990	54433
28	1146	4434	**73**	810	17168	**118**	948	55381
29	600	5034	**74**	290	17458	**119**	1082	56463

30	15	5049	**75**	300	17758	**120**	22	56485
31	36	5085	**76**	529	18287	**121**	75	56560
32	4	5089	**77**	281	18568	**122**	482	57042
33	0	5089	**78**	160	18728	**123**	5509	62551
34	8	5097	**79**	828	19556	**124**	100	62651
35	227	5324	**80**	1011	20567	**125**	10	62661
36	65	5389	**81**	445	21012	**126**	1071	63732
37	176	5565	**82**	296	21308	**127**	371	64103
38	58	5623	**83**	1755	23063	**128**	790	64893
39	457	6080	**84**	1064	24127	**129**	6150	71043
40	300	6380	**85**	1783	25910	**130**	3321	74364
41	97	6477	**86**	860	26770	**131**	1045	75409
42	263	6740	**87**	983	27753	**132**	648	76057
43	452	7192	**88**	707	28460	**133**	5485	81542
44	255	7447	**89**	33	28493	**134**	1160	82702
45	197	7644	**90**	868	29361	**135**	1864	84566
						136	4116	88682

5.2 Results

The proposed model has been applied to the above time between failures dataset and measured the failure rate, probability density failure, mean time to failure and reliability. The values of the N and ϕ, used for further calculation, are calculated using Maximum Likelihood Estimation.

First part of the figure 5.1 depicts the failure rate with respect to the faults. It is clear from the figure that the failure rate is initially high but slide down with the enhance in the failures. It become constant over a period of time and tends to zero at the end. Second part displays the probability density function of the faults. Initially PDF is higher when the software contains maximum faults but decreases as the faults are removed and tends to zero with no faults in the software. However, the third figure explains the reliability of the dataset.

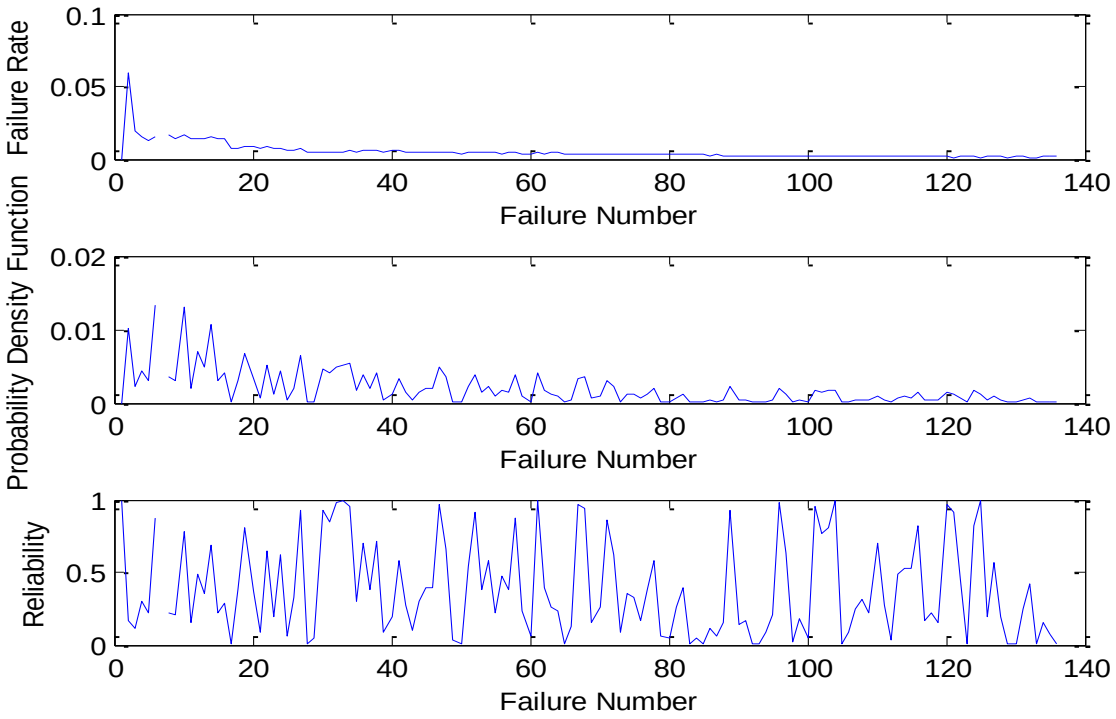

Figure 5.1 Failure Number Vs Failure Rate, PDF and Reliability

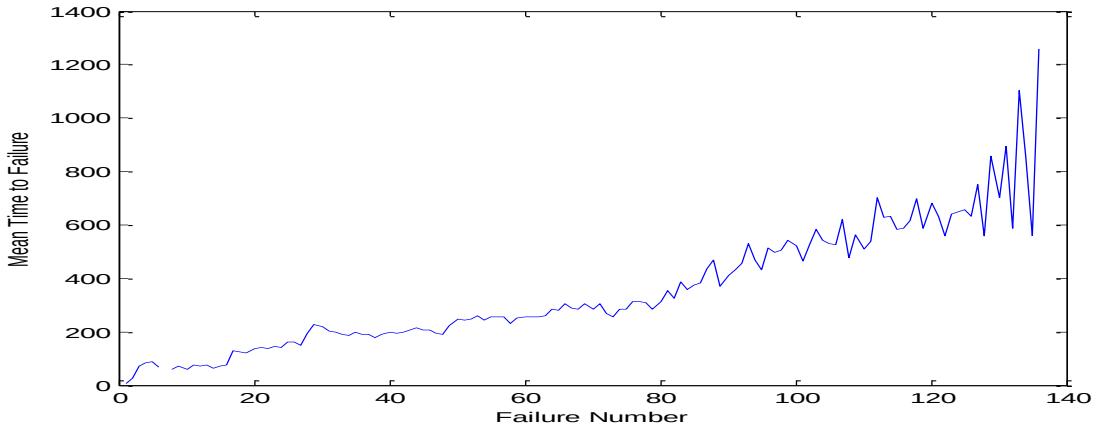

Figure 5.2 Failure Number Vs Mean Time to Failure

The above figure 5.2 depicts the mean time between failures, which is inversely proportional to the failure rate. The MTTF increases with increase in the faults.

5.3 Comparison

The proposed model has been compared with the existing Jelinski-Moranda model and the Goel-Okumoto Imperfect Debugging Model. The Jelinski-Moranda Model, which is very much optimistic model, provides the better results than the proposed model. However, Goel -

Okumoto Imperfect Debugging provides intermediary results, which takes probability as a fixed value which is near to one. The proposed model considered the mean of the probability generated as a random number between 0 and 1.

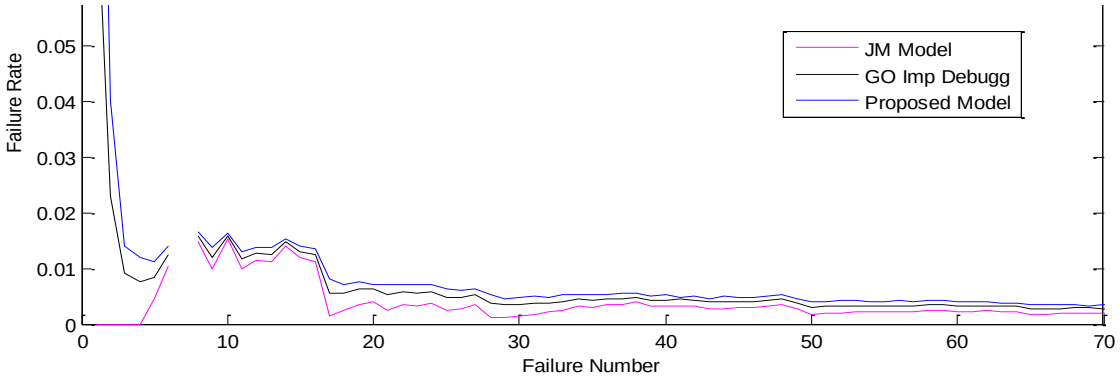

Figure 5.3 Failure Number Vs Failure Rate of Jelinski-Moranda Model, GO Imp Debugg Model and Proposed Model

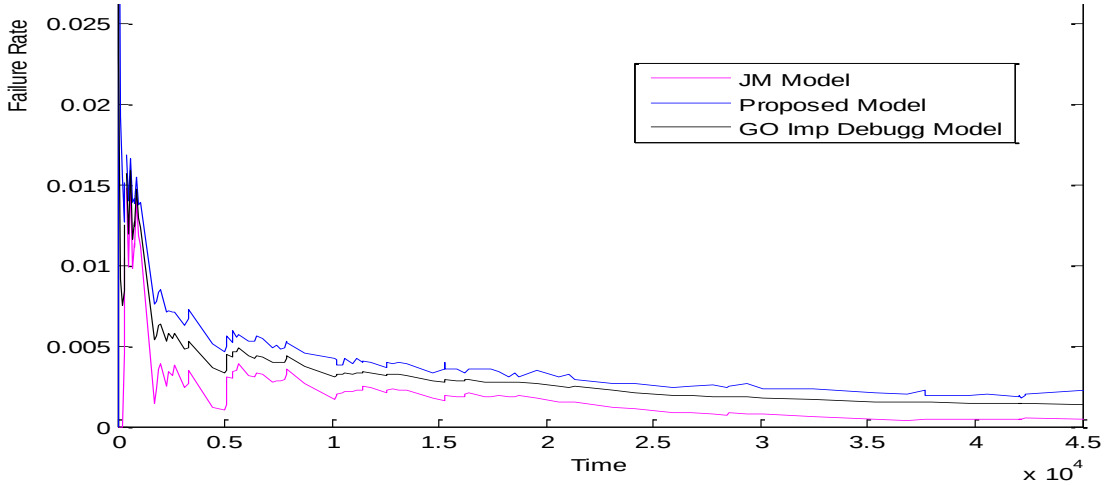

Figure 5.4 Cumulative Time Vs Failure Rate

It is seen from the above figure 5.3, which shows the failure rate with respect to failure number, whereas figure 5.4 shows the failure rate with respect to the cumulative time of failure. The comparison has been carried-out with Jelinski-Moranda Model and Goel-Okumoto Imperfect Debugging model and observed that the results are poor as we considered the impact of the fault being corrected to the other faults.

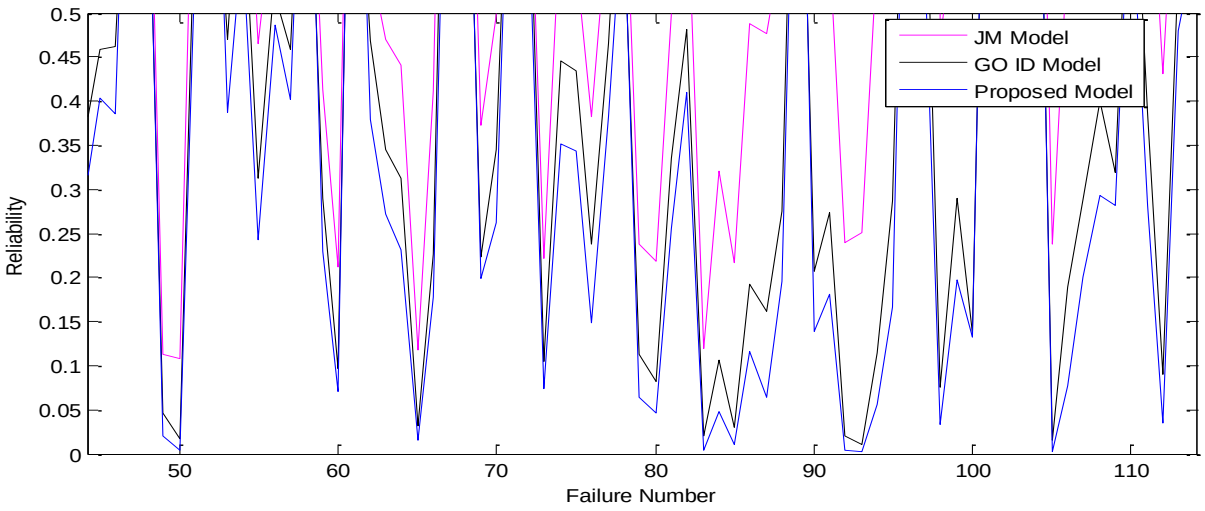

Figure 5.5 Failure Number Vs Reliability of JM Model, GO Imp Debugg Model and Proposed Model

It is observed from the figure that the proposed model is poor reliability graph compare to the other two models, but the assumptions considered are practical and near to the real world situations , instead of highly optimistic as given in Jelinski-Moranda Model. The results of Jelinski-Moranda Model, which considers the fault to be removed perfectly, have better reliability results. Whereas, Goel-Okumoto model has considered that the fault is removed with the probability p, which is near to 1 and depicts the results close to Jelinski-Moranda model, but the results are almost similar to Jelinski-Moranda model with constant difference. However, this is not true in the case of proposed model. The value of the p is generated randomly between 1 and 0 for the N faults and then the mean of the randomly generated probabilities is taken. It is learn from the results or iterations that the mean value of p is around 0.5, which gets changed for each iteration. Hence the results are not same or at the constant difference.

The proposed model will work as the Goel-Okumoto Model, if the value of the p is considered constant and as Jelinski-Moranda Model, if the p is 1 i.e. error is removed perfectly. It is therefore concluded that the Jelinski-Moranda Model and Goel-Okumoto Model are the special case of the proposed model.

CHAPTER-VI

Conclusion and Future Scope

6.0 Conclusion

The results of the proposed model have been analyzed in detail and compared with the existing Jelinski-Moranda Model, which is an extremely optimistic model. Following three of the existing unrealistic assumptions have been changed to new assumptions in the proposed model.

- The fault removal process is not perfect, which means the fault is removed with the probability p, where $0 \leq p \leq 1$.
- Faults are dependent on each other, i.e. the correction of one fault may correct other faults too.
- In the software, each fault contributes differently on the failure, which means the fault of repeatedly executed code improves more reliability compare to the rarely executed codes.

It is observed from the results that the proposed model doesn't have better results and measures less value of the reliability compare to Jelinski-Moranda Model, because the proposed model considers the actual conditions of software development and testing environment. It is concluded from the above discussion that the proposed model is more realistic than the Jelinski-Moranda Model.

6.1 Future Scope

The proposed model can be further extended by considering new fault introduction as a side effect of fault removal process. The probability of fault removal can be better estimated by considering the internal structure of the software and its modules, instead of considering a randomly generated number. Further it can be extended by changing the assumption that the faults are removed immediately, which is impractical in real time environment.

References

1. Bittanti, S. (1988b). A flexible modelling approach for software reliability growth. *Software Reliability Modelling and Identification*.
2. Boehm, B. (1991). *Software Reliability Handbook.* P Rook Elsevier Applied Science.
3. G. S. Mahapatra, P. R. (2012). Modified Jelinski-Moranda Software Reliability Model with Imperfect Debugging Phenomenon. *International Journal of Computer Applications*.
4. Goel, A. (1985). Software Reliability Models: Assumptions, Limitations and Applicability. *IEEE Transaction on Software Engineering*.
5. Goel, A., & Okumoto, K. (1979). Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. *IEEE Transaction on Reliability*.
6. Goel, A., & Okumoto, K. (1979b). A Markovian model for reliability and other performance measures of software systems. *Computer History Museum*.
7. Gokhale, S. (2004). Software Failure Rate and Reliability Incorporating Repair Policies. *Proceedings of the 10th International Symposium on Software Metrics.* IEEE.
8. H. Joe, N. R. (1985). On the Software Reliability Models of Jelinski-Moranda and Littlewood. *IEEE TRANSACTIONS ON RELIABILITY*.
9. Ho, T., Chan, W., & Chung, C. (1991). A quantum modification to the Jelinski-Moranda Software Reliability Model. *IEEE.*
10. Huang, C., Lyu, M., & Kuo, S. (2003). A Unified Scheme of Some Nonhomogenous Poisson Process Models for Software Reliability Estimation. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*.
11. IEEE Standard 729. (1983). Glossary of Software Engineering Terminology. *IEEE*.
12. Jelinski, Z., & Moranda, P. (1972). Software Reliability Research. *Statistical Comnputer Performance Evaluation.*
13. Kapoor, P., Shatnawi, O., & Aggarwal, G. (2009). Unified Framework for Developing Testing Effort Dependent Software Reliability Growth Models. *Wseas Transactions on Systems.*
14. Kharchenko, V., Tarasyuk, O., Sklyar, V., & Dubnitsky, V. (2002). The Method of Software Reliability Growth Models Choice Using Assumptions Matrix. *Annual International Computer Software and Applications Conference.* IEEE.
15. Littlewood, B. (1981). Stochastic Reliability-Growth: A Model for Fault-Removal in Computer-Programs and Hardware-Designs. *IEEE Transaction on Reliability*.
16. Littlewood, B., & Sofer, A. (1987). A Bayesian modification to the Jelinski-Moranda software reliability Growth Model. *Software Engineering Journal*.
17. Littlewood, B., & Verral, J. (1979). *IEEE Symposium on Software Model*.

18. Liu, Z., Zhang, R., Mei, D., & Zheng, X. (2011). An Improved Software Reliability Model Considering Fault Correction Rate. *International Conference on Computational Intelligence and Security.* IEEE.

19. Lyu, M. (2002). Software Reliability Theory. *Encyclopedia of Software Engineering*.

20. Mellor, P. (1987b). Software reliability modelling: the state ofthe art. *Information and Software Technology*.

21. Musa, J. (1987). Software Quality and Reliabitity Basics. *IEEE*.

22. Musa, J., & Okumoto, K. (1984). A Logarithmic Poisson Execution Time Model for Software Reliability Measurement. IEEE.

23. Musa, J., Iannino, A., & Okumoto, K. (1987). *Software Reliability Measurement Prediction Application.* McGrawHills.

24. Myung, I. (2002). Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*.

25. Nakagawa, Y. (1994). A connective exponential software reliability growth model based on analysis of software reliability growth curves. *IEICE Transactions*.

26. Ohba, M. (1984). Software reliability analysis models. *IBM Journal of Research Development*.

27. Ohba, M., & Yamada, S. (1984b). S-shaped software reliability growth models. *Int.Conf. Reliability and Maintainability.*

28. Peng, R., Hu, Q., & Ng, S. (2008). Incorporating Fault Dependency and Debugging Delay in Software Reliability. *IEEE ICMIT.* IEEE.

29. Pham, H. (2000a). Software Reliability. *Springer-Verlag*.

30. Pham, H. (2006). *System Software Reliability.* New Jersey: Springer.

31. Pham, H., & Hwang, S. (2009). Quasi-Renewal Time-Delay Fault-Removal Consideration in Software Reliability Modeling. *IEEE Transactions on Systems, Man, and Cybernetics*.

32. Pham, H., & Normann, L. (1997b). generalized NHPP software reliability model. *ISSAT International Conf. on Reliability and Quality in Design.*

33. Pham, H., Nordman, L., & Zhang, X. (1999). A General Imperfect-Software-Debugging Model with S-Shaped Fault-Detection Rate. *IEEE Transaction on Reliability*.

34. Pham, H., Nordmann, L., & Zhang, X. (1999b). A general imperfect software debugging model with s-shaped fault detection rate. *IEEE Transactions on Reliability*.

35. Pressman, R. (2001). *Software Engineering: a practitioner's approach.* The McGraw-Hill Companies, Inc.

36. Quyoum, A., Dar, M., & Quadari, S. (2010). Improving Software Reliability using Software Engineering Approach- A Review. *International Journal of Computer Applications*.

37. Ramamoorthy, C., & Bastani, F. (1986). Input-domain-based models for estimating the correctness of process control programs. *Reliability Theory*.

38. Ramoorthy, C., & Batsani, F. (1979). A systamatic Approach to developmentand validationof critical software for nuclear power plant. *IEEE*. IEEE.

39. Razeef, M., & Nazir, M. (2012). Software Reliability Growth Models: Overview and Applications. *Journal of Emerging Trends in Computing and Information Sciences*.

40. Rinsaka, K., & Dohi, T. (2004). Who Solved the Optimal Software Release Problems Based on Markovian Software Reliability Model. *IEEE International Midwest Symposium on Circuits and Systems.* IEEE.

41. Schick, G., & Wolverton, R. (1978). (1978). An analysis of competing software reliability models. *IEEE Trans. Software Eng.*

42. ShanthiKumar, J. (1981). A general software reliability model for performance. *Microelectronics and Reliability*.

43. Sharma, K., & Garg, R. (2011). *Software Reliability Modeling and Selection.* LAMBERT Academic Publishing.

44. Sharma, K., Garg, R., & Nagpal, C. (2010). Selection of Optimal Software Reliability Growth Models Using a Distance Based Approach. *IEEE Transactions on Reliability*.

45. Sukert, A. (1977). An investigation of software reliability models. *Proc. Ann. Reliability and Maintainability Symp.*

46. Trivedi, K. (2001). *Prboability and statistics with reliability, queuing and computer science applications.* Durham, North Carolina: Wiley-Interscience Publication.

47. Wood, A. (1996). *Software Reliability Growth Models.* CA: Tandem.

48. Xie, M. (1991). *Software Reliability Modelling.* World Scientific Publishing Co. Pte. LId.

49. Xie, M., & Bergman, B. (1988). On modelling reliability growth for software. *8th IFAC Symp. on Identification and System Parameter Estimation.* Beijing, China.

50. Yamada, S., & Osaki, S. (1985). Software reliability growth modeling: models and applications. *IEEE Transactions on Software Engineering*.

51. Yamada, S., Ohba, M., & Osaki, S. (1983). S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability*.