CHAPTER 1
SOFTWARE RELIABILITY-SOME PRELIMINARIES

1.1 Introduction

Some of the most fundamental forces which have shaped the twentieth century are technological in origin and relate to transportation, communication, energy and manufacturing technology etc. Advances in these areas are linked to the invention and widespread use of automobiles and airplanes, radio, television and radar; nuclear and solar energy; and the assembly line and automated manufacturing. Two other developments interact with and magnify the impact of all the above mentioned colossal inventions, are electronics and the computer. Furthermore, the last two are inseparably linked, since our modern computers are electronic, digital devices.

While a computer system has input, output, memory and processing components, the software is based on algorithms, programs and documentation. Every engineer and scientist know that there are periods in technological development when state-of-the-art limitations or economic constraints restrict what one can do. However, when it comes to the computer, in a majority of cases, we have been successful, and this has led to leaps of technological advancement along with some level of unreliability also, which outweighs the benefits.

A computer software is required to instruct the computer hardware in performing the necessary computations. An important goal is to learn how to initially estimate and subsequently measure the quality, reliability of the end product. The technique of design in most engineering fields is not one of synthesis, but one of iteration. An approximate design is conceptualized and then analyzed to see if it meets the specifications; if not, the design is modified. This process is iterated a number of times to yield a satisfactory design. Thus, inherent in the design process is an ability
to analyze the design and then compare the results of the analysis with the desired specifications.

A software system has to pass through a number of iterations before shaping itself into a desirable product. This is a distinguishable feature of software product from other manufactured products where the number of iterations are not as many as that of software product. In the ultimate stage what is required is a reliable and better quality product. For a software product also the methodologies of quality control and reliability can be applied.

1.2 Software Quality and Reliability

Quality is defined as a desirable characteristic of something and quality control can be said as a series of instructions, reviews and tests used throughout the development of products to ensure that each work product meets the requirements placed upon it. In order to be competent enough in software industry there has to be a reference for quality. The growth of industry depends directly on the quality that can be achieved if it is implemented in all phases of software development. Lots of strategies are adopted for quality like prevention of defects, minimizing the number of bugs so that the quality requirements are met. Certain standards are to be maintained in areas like future design, programming, integration tests, system testing etc.

Since softwares are intellectual creation, it is difficult generally to measure the software quality. However it can be judged by its correctness, adaptability, testability, obsolescence and intraoperability. In addition to this a software product has a typical development process as some activities are related to particular phases of development and the other activities of the software apply throughout the software development. As in the case of any other product the only way for a software quality is to identify systematically the opportunities of improvement at every stage of software development.
An important quality attribute of a computer system is the degree to which it can be relied upon to perform its intended functions. Evaluation, prediction, and improvement of this attribute have been of concern to designers and users of computers from the early days of their evolutions. Software is essentially an instrument for transforming a discrete set of inputs into a discrete set of outputs. It comprises a set of coded statements whose function may be to evaluate an expression and store the result in a temporary or permanent location, decide which statement to execute next, or to perform input/output operations.

Since, to a large extent, software is produced by humans, the finished product is often imperfect. It is imperfect in the sense that a discrepancy exists between what the software can do versus what the user or the computing environment wants it to do. These discrepancies are what we call software faults. Basically, software faults can be attributed to an ignorance of the user requirements, ignorance of the rules of the computing environment, and to poor communication of software requirements between the user and the programmer or poor documentation of the software by the programmer. Even if we know that software contains faults, we generally do not know their exact identity. There are two approaches available for indicating the existence of software faults, namely program proving, and program testing.

In practice neither proving nor testing can guarantee complete confidence in the correctness of a program. Each has its advantages and limitations and should not be viewed as computing tools. They are, in fact, complementary methods for decreasing the likelihood of program failure. Due to the imperfectness of these approaches in assuring a correct program, a metric is needed, which reflects the degree of program correctness and which can be used in planning and controlling additional resources needed for enhancing software quality. One such quantifiable metric of quality that is commonly used in software engineering practice is software reliability. A commonly used approach for measuring software reliability is via an analytical model whose parameters are generally estimated from available data on software failures. Reliability and other relevant measures are then computed from the fitted model.
There are a number of views as to what software reliability is and how it should be quantified. Some people believe that this measure should be binary in nature so that an imperfect program would have zero reliability while a perfect one would have a reliability value of one. This view parallels that of program proving whereby the program is either correct or incorrect. Others, however, feel that software reliability should be defined as the relative frequency of the times that the program works as intended by the user. This view is similar to that taken in testing where a percentage of the successful cases are used as a measure of program quality.

Software reliability is a probabilistic measure and can be defined as the probability that software faults do not cause a failure during a specified exposure period in a specified use environment. The probabilistic nature of this measure is due to the uncertainty in the usage of the various software functions and the specified exposure period, that may mean a single run, a number of runs, or time expressed in calendar or execution time units.

Software reliability is a useful measure in planning and controlling resources during the development process so that high quality software can be developed. It is also a useful measure for giving the user confidence about software correctness. Planning and controlling the testing resources via the software reliability measure can be done by balancing the additional cost of testing and the corresponding improvement in software reliability. As more and more faults are exposed by the testing and verification process, the additional cost of exposing the remaining faults generally rises very quickly. Thus, there is a point beyond which continuation of testing to further improve the quality of software can be justified only if such improvement is cost effective. An objective measure like software reliability can be used to study such a tradeoff.

Hardware exhibits mixtures of decreasing and increasing failure rates. The decreasing failure rate is seen due to the fact that, as test or use time on the hardware system accumulates, failures most likely due to design errors, are encountered and
A number of analytical models proposed to address the problem of software reliability measurement can be classified into four main groups, according to the nature of the failure process studied. They are times between failure models, failure count models, fault seeding models and input domain based models.

**Times between failure models:**

In this class of models the time between failures is a random variable following a certain probability distribution. Specifically if $T_i$ denotes the time between $(i-1)^{t}$ and $i^{th}$ failure, modeling will be done on the random variable $T_i$. The parameters of the distribution of $T_i$ are assumed to depend on the number of faults remaining in the software during the time $T_i$. The distribution of $T_i$ reflects the improvement in the software quality as faults are detected and removed from the software. The basic model of this category is due to Jelinski and Moranda (1972) given by

$$Z(t_i) = \phi[N-(i-1)]$$

where $t_i$ is a realization of $T_i$

$Z(t_i)$ is the software failure rate

$N$ the total number of faults in the system at the start of testing

$\phi$ is constant of proportionality
Some modifications of this model are subsequently given by Schick and Wolverton (1973), Goel and Okumoto (1978). The model that falls in this category but suggested through Bayesian approach is that of Littlewood and Verral (1973). Another model of this category through notion of geometric deeutrophication is by Moranda (1975). Estimates of the parameters of this class of models are obtained from observed values of times between failures in order to get the estimates of software reliability, mean time to next failure etc.,

**Failure count models:**

The interest of this class of models is the number of failures in a specified time interval rather than times between failures. The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. As faults are removed from the system, it is expected that the observed number of failures per unit time will decrease. Hence the cumulative number of failures versus time will eventually level off. The time metric here can be calendar time, CPU time or number of test cases run etc. The time intervals are fixed and observed number of failures in each interval is treated as a random variable. The basic idea behind most of these failure count models is that of a Poisson distribution whose parameter takes different form for different models. Letting \( N(t) \) be the cumulative number of failures observed by time \( t \), \( N(t) \) can be modeled as a Poisson process with a time dependent failure rate.

\[
P[N(t) = y] = \frac{[m(t)]^y e^{-m(t)}}{y!}, y = 0, 1, 2, \ldots
\]

Here \( m(t) \) is the expected number of failures observed by time \( t \). It should be noted that here the number of failures to be detected is treated as a random variable whose observed value depends on the test and other environmental factors. This is a fundamental departure from the other models which treat the number of failures to be a fixed unknown constant. Different forms of the function \( m(t) \) might be more suitable in different environments. These models are named as Non-homogeneous
Poisson process (NHPP) models of the category failure count models. The pioneering models of this category are due to Goel and Okumoto (1979), Goel (1982,1983), Yamada et al (1983) and Musa and Okumoto (1984).

Fault seeding models:

This is a separate variety of models in which a known number of faults is placed in the program. After testing, the numbers of exposed seeded and indigenous faults are counted. Using combinatorics and maximum likelihood estimation, the number of indigenous faults in the program and reliability of the software can be estimated. The most basic fault seeding models are due to Mills (1972), Lipow (1972) and Basin (1974).

Input domain based models:

In this approach a set of test cases from an input source is generated. These cases cover the operational profile of the input source, where the source is partitioned into a finite number of classes each of which is associated with a path of the program. The reliability of the software is calculated from the number of failures observed during the execution of the selected test cases. Some works related to input domain models are by Brown and Lipow (1975), Nelson (1978), Ramamurthy and Bastani (1982) and Bastani (1985).

In all the above categories of the models it is helpful to note a precise statement of the assumptions required to develop the model. Some of the commonly stated assumptions to develop a software reliability model are:

1) Times between failures are independent.
2) A detected fault is immediately corrected.
3) No new faults are introduced during the fault removal process.
4) Failure rate decreases with test time.
5) Failure rate is proportional to the number of remaining faults.
6) Reliability is function of the number of remaining faults.
7) Time is used as a basis for failure rate.
8) Likelihood of finding a fault increases as the testing time increases within a given failure interval.
9) Testing is representative of the operational use.

With the above listed general assumptions the four categories of reliability models are specified to have been developed with the key assumptions for the respective model category as given below:

**Times between failures (TBF) models.**
- Independent times between failures.
- Equal probability of the exposure of each fault.
- Embedded faults are independent of each other.
- Faults are removed after each failure occurrence.
- No new faults are introduced during correction that is, perfect fault removal.

**Failure count (FC) models.**
- Testing intervals are independent of each other.
- Testing within intervals is reasonably homogeneous.
- Number of faults detected during non-overlapping intervals are independent of each other.

**Fault seeding (FS) models.**
- Seeded faults are randomly distributed in the program.
- Indigenous and seeded faults have equal probabilities of being detected.

**Input domain based (IDB) models.**
- Input profile distribution is known.
- Random testing is used.
- Input domain can be partitioned into equivalent classes.
In the above four categories of models our particular interest is about those which
deal with the random number of software failures in a given period of time or the
inter failure time of a developed software. With these two concepts as the focal theme
we propose to study the quality and reliability of a software product.

1.3 Software Reliability Growth Models

Counting process models have played a key role in the analysis of software
failure data. Suppose that we are interested in observing the occurrences of a
repeatable event over a period of time. An example that is of interest to us here is the
points in time at which a software fails. Such events do not occur with any regularity
and are therefore unpredictable. Consequently we are not sure about the times at
which the event will occur and also about the number of events that will occur in any
time interval. A counting process is a count of the number of events that have
occurred in a specified interval of time. In notation we may represent it by \( N(t) \) which
is unknown aprori for any value of \( t \geq 0 \). Hence we are faced with the problem of
describing uncertainty about an infinite collection of random variables one for each
value of ‘t’. Such a collection is called a stochastic counting process denoted by \([N(t),
t \geq 0 \) ]. If future changes in \( N(t) \) are independent of the past changes in it, the
probability distribution of \( N(t) \) in a time interval say \([t_1, t_2] \) depends only on the
length \( t_2-t_1 \) but not on the extremities \( t_1, t_2 \), as a constant multiplier of \( (t_2-t_1) \) say
\( m(t_2-t_1) \), then the probability distribution of \( N(t) \) for any non negative ‘t’ can be
derived as

\[
P \left[ N(t) = y \right] = \frac{e^{-mt}(mt)^y}{y!}, \quad y = 0, 1, 2, \ldots, m > 0.
\]

This is called a homogeneous Poisson process (HPP). We know that \( E[N(t)] = mt \),
called the mean value function of the HPP. A Poisson process model for describing
uncertainty about the number of software failures in a given time \((0, t)\) is given by the
probability equation.
\[ P[N(t) = y] = \frac{e^{-m(t)}[m(t)]^y}{y!}, \quad y = 0, 1, 2, \ldots \]

Where \( m(t) \) is a finite valued non negative and non decreasing function of 't' called the mean value function. Such a probability model for \( N(t) \) is said to be an NHPP model. The derivative of \( m(t) \) is called the intensity function of the NHPP. Various software reliability growth models are proposed in the literature, assuming different forms for \( m(t) \). This helps us to get the software reliability of the system in the interval \([t, t+x]\) as given by

\[ R(x/t) = e^{-[m(t+x)-m(t)]} \]

Having introduced these concepts of software reliability we can say that the notion of a quality software is governed by a random phenomenon in a number of ways. Two such situations are the number of experienced failures of a software in a given period of time during testing/operational phase which can be described by a time dependent discrete random process and another situation is the continuous time random variable representing the lapse of time between two consecutive failures of a software. In either case the preferability or otherwise of a software can be assessed by the observable data on the above two random phenomena. Effectively this is equivalent to studying the probabilistic nature of an observable software failure data and drawing inferential conclusions out of such a study. Many authors have been working in similar research areas. We now present a detailed survey of the research literature dealing with probabilistic aspects of software reliability in a chronological order. Though we have put up our best efforts in presenting this literature, we cannot rule out any possible omissions in this regard. However this survey helped us to trace out some research problems for further investigation that form our research activity of this thesis.

1.4 Research Literature and Proposed Study

Trivedi and Shooman (1975) suggested a many state Markov model for the purpose of estimation and prediction of software performance parameters. The
availability and reliability of the software system are also presented for the purpose of predictions. Based on the notion of the distribution function of random variable, Forman and Singpurwalla (1977) developed a probabilistic model describing the software failure phenomenon to suggest estimates of the parameters in the model and termination procedure for debugging the software. Schick and Wolverton (1978) describes the most commonly used software reliability growth models as divided into two groups of time domain and data domain. They present a comparative picture of the model as applicable to various actual data sets. Goel and Okumoto (1979) are two of the foremost researchers that have considered the probabilistic nature of software failure phenomenon based on an NHPP. They have analyzed the failure process to develop a suitable mean value function which in turn is used to get software performance measures like distribution of the cumulative number of software errors, the residual number of errors, software reliability, maximum likelihood estimators of the parameters of the model and joint confidence region of the parameters. The mean value function of this model turns out to be based on the cumulative distribution function of the well-known exponential distribution. Musa (1980) presented the need for potential use of software reliability measurement and made a comparison of software and hardware reliabilities. The concepts of software reliability, software and hardware components availability are also presented. Ramamurthy and Bastani (1982) reviewed the status and perspectives of software reliability as on 1982. Iannino et al., (1984) gave a descriptive narration of various criteria for the comparison of software reliability models based on predictive validity, capability, quality of assumptions, applicability, simplicity.

Singpurwalla and Soyer (1985) suggested random coefficients auto regressive process to assess software reliability growth and applied their approach to a real life time data. Matsumoto et al., (1988) discuss the evaluation procedure of a SRGM using data from a single program testing process applied to exponential, hyper exponential and S-shaped models, ranking the S-shaped model as superior with respect to estimation as well as goodness of fit. A general model that fits into empirical Bayes of software reliability is suggested by Mazzuchi and Soyer (1988).
Crow and Basu (1988) studied reliability estimation with incorrectly reported data using ML estimation for the U.S. Army Material Systems Analysis Activity (AMSAA) model. Mazzuchi et al., (1989) presented an overview of the proportional hazards model of the survival analysis and discussed its use in assessing software reliability. Tohma et al.,(1989) presented applicability of hyper geometric distribution to estimate residual number of software faults to real test and debug data. Gibson and Crow (1989) developed a methodology for estimating average fix effectiveness factor in a reliability growth test phase. Fault density and failure intensity of an SRGM are used as two metrics to monitor software development capabilities and to measure customer satisfaction with the developed product in the research investigation of Huensch et al. (1990). Ehrlich et al.,(1990) used the software reliability data collected during the testing of a system to measure the software quality in terms of experienced software failures. The notions of exponential NHPP are made use of for goodness of fit of the data. Csenki (1990) presented the use of Bayes prediction analysis to derive the predictive distribution of time to next failure. The concept is applied to the Jalenski-Moranda software reliability model. The software reliability growth, decay in a given period of software failure data specifically a software production concern is evaluated by Bastos et al.,(1990) using Laplace trend analysis for exponential and S-Shaped growth models. Jacoby and Tohma (1990) used hyper geometric distribution to estimate the number of initial faults in a software at the beginning of the test and debug phase. The time to repair, mean logistic delay time variates at the system level are modeled by Crow (1990). Tohma et al., (1991) investigated six ways of the estimation of parameters in a hyper geometric distribution to get an estimate of number of initial faults in a program at the beginning of its testing/debugging, along with their relative accuracies. Sofer and Miller (1991) use a nonparametric method of estimating the software failure rate in completely monotone models which can be compared with parametric approaches. Shooman (1991) discusses a micro software reliability model that can be used to apportionment of reliability and test efforts among the various execution paths. Vallee and Ragot (1991) demonstrated the application of NHPP approach to the industrial world generating accurate predictions with specific applications in space research. Ehrlich
et al., (1991) analyzed individual per fault failure sequence resulting from delayed fault detection and software correction during test execution to conform the applicability of a Poisson process. Lyu and Nikora (1991a) proposed linear combinations of software reliability models for the purpose of automating the procedures of software reliability analysis. The concept of equally weighted linear combination model resulting from linear combination of three popular software reliability models is suggested by Lyu and Nikora (1991b) for applying to software failure data sets. Tohma et al., (1991) made use of hyper geometric distribution to estimate by least squares sum method the number of faults initially present in a program at the beginning of test/debug process arriving at reduction in the time needed for calculating the estimates. Jacoby and Tohma (1991) explained the idea of capture-recapture process for software faults in the context of a proposed testing environment. Estimation of the degree of unavailability of the software system are also given for hyper geometric distribution. Wing et al., (1991) studied the effect of redundancy on the requirements of logistic support of an SRGM. An NHPP based SRGM influenced by error removal phenomenon is suggested by Kapur and Garg (1992) along with its predictive validity. With the help of curve fitting techniques predicting the number of faults in a system through fitting non-linear regression models to the number of faults in a program module, an SRGM is suggested by Khoshgoftaer et al., (1992). The choice of an appropriate software reliability model on the basis of a predictability measure is proposed by Malaiya et al.,(1992) and applied to logarithmic model, inverse polynomial model and delayed S-shaped model. The performance of software reliability models is quantified and compared with respect to a measure that helps in preferring one model over the other by Downs and Scott (1992). A new software reliability model called the log power NHPP SRGM is proposed by Zhao and Xie (1992), which has relatively high predicting ability and is simple for graphical interpretation. Gaudoin (1992) studied the use of the Laplace trend test to detect software reliability growth and applied its usefulness for various SRGMs. Jacoby and Masuzawa (1992) made use of hyper geometric distribution model in software with test coverage.
If testing and operation are the two phases of a developed software, taking into considerations the notion of software reliability during testing phase an SRGM is suggested by Yamada et al., (1993). This model incorporates the amount of test effort as a random variable following a Weibull distribution. An SRGM useful for operational use is suggested by Kenney (1993) that can estimate the residual number of defects, anticipated arrival times of customer reported failures taking into the concept of “power function of time” associated with Weibull distribution. The concept of data aging is made use of to identify the starting failure count of a software failure data in an optimum way to choose an SRGM for failure data analysis by Schneidewind (1993). This model suggests four optimal criteria to determine the start of a failure count. Necessary and sufficient conditions for the existence and finiteness of the MLEs of the parameters of SRGMs are derived by Hussain and Dahiya (1993) and applied to various standard models. Confidence interval procedures for power law based on NHPP SRGM are suggested by Crow (1993). Gaudoin et al., (1994) generalized the concept of quantified quality debugging of Gaudoin and Soler (1992) from deterministic to probabilistic and suggested another model called log normal proportional model as software reliability model. Based on the concept of different rates of debugging at different stages of a software development, Kececioglu et al., (1994) worked out a reliability growth model using the well known Gompertz curve which is a special fit for failure data with S-shaped trends. Sylvia and Singpurwala (1994) propose a Bayesian approach for predicting the number of failures in a software using the log Poisson execution time model based on published information on the empirical experiences of other researchers. They have named such an information as expert opinion. Whittaker and Thamson (1994) described a method for statistical testing based on a Markov chain of model of software usage with a two fold significance. The influence of failure is assessed and a stoppage rule for the testing process is derived.

The two main characteristics of a software development process are:

(i) No programmer is perfect and thus when an error is removed new errors can be introduced into the program.
(ii) Not all errors are created equal. Different errors will have different implications and need different handling.

The first characteristic mentioned above is called imperfect debugging. Some people who worked in these areas are Kuo (1983), Yamada et al., (1984), Yamada and Osaki (1985, 86), Ohoba and Chou (1989), Kareer et al., (1990), Leung (1992), Kapur and Bhalla (1992). Combining the above two characteristics Lynch et al., (1994) developed a software reliability model to determine the optimal debugging time necessary to minimize cost without disturbing reliability constraints. Kapur et al., (1994) incorporated the concept of imperfect debugging with testing effort in a software reliability model that can be used to plan the amount of testing effort required to achieve a predetermined target in terms of errors removed in a given span of time. Zeephongsekul et al., (1994) have introduced an SRGM with imperfect debugging where the mean total number of failures (primary and secondary) and gave a method of MLE to estimate its parameters. Nara et al., (1996) report the suitability and stability of the NHPP and trend curve SRGMs to actual software development projects. Quality control limit curves are also suggested for the effect of application of the models. Wood (1996) has considered a panel of eight SRGMs in order to predict software reliability on the basis of coded data on four releases of a developed software using the principle of least squares. Of the eight models the most popular exponential model-Goel and Okumoto (1979) was the choice for prediction purposes. Confidence intervals that account for any uncertainty concerning the operational profile of a software are presented by Adams (1996). The theory of order statistics of statistical inference is made use of by Mitchell and Zeil (1996) in presenting a software reliability model with a notion of directed testing method. Software reliability prediction in a nonparametric scenario through Bayesian approach is studied by Aroui and Solefr (1996). The optimal testing rule and hence the optimal release policy for a model based on hyper geometric distribution are investigated by Hou et al., (1996). Shima et al., (1997) examined the failure intensity distribution of an SRGM from an empirical failure data and show that the procedure predicts number of detected faults more accurately than the conventional ones. Incorporating the concept of explicit debugging activities along with the possibility of imperfect
debugging, Gokhale et al., (1998) have suggested software reliability analysis with fault detection and debugging. The study is based on the estimates of residual number of faults in the software. The software aging phenomenon is an equally important notion that has to be given some consideration. This is expressed in terms of accumulated errors during the execution of the software. The effect of aging is quantified with the help of a metric called estimated time to exhaustion. These aspects are studied in Garg et al., (1997). The Bayes estimate of mean number of errors present in a software system, the system reliability along with numerical illustrations are given by Kuo et al., (1997). Safety critical analysis applied in reliability modeling in order to increase the trust in the reliability of such products is the contribution of Schneidewind (1997) by integrating software safety criteria, risk analysis, reliability prediction and stopping rule for testing. Cai (1997) considered the non failure stops of software execution process as a type of censored data and developed a reliability model by censoring the existing software reliability models. Pointing out the inadequacy of some SRGMs in describing the failure process of a software failure data, Gokhale and Trivedi (1998) propose the mean value function of an NHPP as given by a log logistic model that can indicate the increasing as well as decreasing nature of the failure occurrence. Assuming that event rates in software systems are generated by multiplicative process, the distribution of software failure rates is modeled by log normal distribution in Mullen (1998) and thereby a software reliability model is constructed, called log normal execution time software reliability growth model. This model is used to analyze two series of failure data and the likelihood of the data arising from lognormal, log Poisson is computed, that revealed favourability to the log normal model. Software reliability modeling through concatenating failure rates to overcome the lack of memory feature of a Poisson process is suggested by Singpurwalla (1998). Thomson and Whittaker (1998) considered the combination of the notions - software failure as rare event and finite state discrete parameter recurrent Markov chain. The results are applied to the analysis of software reliability for systems of high quality. Evanco (1999) used the notion of proportional hazard model in the form of a software development by identifying the three explanatory variables namely software complexity, software
development volatility and cumulative execution time as variables influencing software reliability. He developed the regression approach to effect the reliability of a software by these three variables. Relaxing the nature of statistical independence that is generally assumed among successive software failures, Goseva and Trivedi (2000) suggested software reliability modeling frame work based on Markov renewal process. This modeling becomes a general case of many models that assume failure independence. Keiller and Mazzuchi (2000) investigated the improvement in the performance of Goel-Okumoto model with various smoothing techniques including Laplace trend test. Pham and Pham (2001) studied the predictive performance of a software reliability model inserting pseudo failures based on Bayes approach. Combination of failure detection and fault correction in a single model is done by Schneidewind (2002) in order to have a prediction support. Reliability assessment of software divided into a finite number of models with the help of a Markovian model and that of statistical test of hypothesis is studied by Rajgopal and Mazumdar (2002). Homogenizing failure data with respect to failure intensities and adopting different SRGMs for different failure intensities, a combined SRGM for a better reliability assessment is suggested by Tian (2002). Keiller and Mazzuchi (2002) studied the performance of a set of SRGMs using smoothing techniques including Laplace trend test. Huang and Kuo (2003) describe a unified scheme of estimation in an SRGM using weighted arithmetic mean, weighted geometric mean and weighted harmonic mean. Their unified approach was verified to cover many of the well known NHPP models under different sets of conditions. Boland and Singh (2003) adopted a birth process approach to Moranda geometric software reliability model and derived the mean value function, intensity function, reliability function, probability distribution of relevant point process and optimum release time for the software. A concept similar to imperfect debugging is fault removal efficiency. Combining the notions of fault removal efficiency, failure rate and fault introduction rate a unified model that integrates these three is proposed by Zhang et al., (2003). Grottke and Mullen (2004) describe the concepts of qualified testing in the test coverage in software based on lognormal failure rate model. On the assumption of improvement in reliability along with time irrespective of bugs are corrected or not, a simple software reliability model
is proposed by Jalote and Murphy (2004). The notions of faults dependency and time
dependent delay are introduced by Huang et al., (2004) and described an SRGM
with imperfect debugging. A unified mathematical modeling for explaining the
imperfect debugging in a software fault process by dividing the original failure
process of the model into two different NHPPs is introduced by Lo and Huang
a software tool in short- ReDCAS employing Bayesian techniques to estimate
reliability measures. The concept of infinite time to next failure is the general
consideration of many SRGMs indicating that the software product is too good. A
relaxation of this criterion is the summary of the study by Gokhale and Trivedi (2005)
to give a method for mending time to failure distributions. Mullen and Gokhale
(2005) have considered a Poisson process with log normal distribution as the mean
value function and presented goodness of fit tests for software failure data. The
estimates of software failure rates adjusted in tune with the test data for the suitability
of appropriate reliability models is considered by Jeske et al., (2005). An efficient
software reliability assessment model for the actual open source system development
to compare between S-shapeć and other models is investigated by Tamura and
Yamada (2005). Ohishi et al., (2005) proposed the NHPP based on Gompertz curve
as a software reliability model and applied it to a live data. Keiller and Mazzuchi
(2005) suggested some methods of comparing software reliability models using
smoothing techniques. Crow (2005) gave a detailed note about the methods
improving the effectiveness of the reliability tasks. The concept of multiple change
points and imperfect debugging are integrated in a single model to study the software
reliability by Huang and Lin (2005)....

We know that software reliability is probability that the software errors induced
by faults do not occur for a specified time period. Hence the time evaluation of error
occurrence process is modeled by stochastic counting processes. When it is not
always easy to monitor the software test execution, we have to depend on the number
of test runs as an assignable cause to know about the number of software errors
experienced. In other words we count the number of software errors experienced in
the testing phase at a discrete calendar time like hour, day and week. In such situations, discrete software reliability models will be useful to assess the reliability of software product. Such models are proposed by Yamada et al. (1984, 85). Other works on discrete software reliability models are Scholz (1986), Boland et al., (1987), Kapur et al., (1994), Goseva-Pospojanova and Trivedi (2000), Satoh and Yamada (2001), Okamura (2005), Tang and Pham (2006).

After thoroughly scanning the literature presented in Section 1.4, we are motivated to study the following research problems related to software reliability.

(i) Based on the notion of an expected decreasing failure intensity as time advances, online of Musa and Okumoto (1984), we propose a software reliability growth model with practical relevance, and study the reliability assessment with respect to the proposed model.

(ii) As mentioned in the survey of literature, combination of SRGMs are some times more relevant. We take up some pooled SRGM as our another problem of research investigation. While these proposals are regarding estimation / prediction of software reliability through model building, our interest got inclined to the work of Stieber (1997) that deal with detection of unreliable software components using the theory of Wald’s sequential probability ratio test (SPRT) procedure as a statistical quality control tool. It is about a homogenous Poisson process. As no further publications on lines of Stieber (1997) are traced out by us, we attempt to adopt the methodology of Stieber (1997) to an NHPP with a general mean value function and specialize it for three specific NHPPs of which two are quite popular in literature and one is our proposed model. Our attempts and findings in these directions are presented in the chapters that follow with an aim of proposing analytical techniques for quality software.