2.1 Introduction

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

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

2.2 Reliability Growth Models

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

A variety of models are available for the estimation of reliability in the case of software as well as hardware. The role that software plays as a support to modern societal activities cannot be underestimated. However,
the ability to predict software reliability is still not well understood and it needs further study. Although a number of software reliability models have been developed till date, none has been universally accepted in the field (Smidts and Li [2002], Li and Smidts [2003]).

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

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

A software reliability growth model provides a systematic way of assessing and predicting software reliability based on certain assumptions about the fault in the software and fault exposure in a given usage environment (Joe et. al [1993]). The reliability growth for software is the positive improvement of software reliability over time, accomplished through the systematic removal of software faults. The rate at which the reliability grows depends on how fast faults can be uncovered and removed. A software reliability growth model allows project management to track the progress of the software’s reliability through statistical inference, and to make projections of future milestones (Lakey [1997], Musa and Okumoto [1983]). Models are classified in terms of five different attributes. Time domain: Wall clock
versus Execution time. Category: Total number of failures that can be experienced in finite or infinite time. Class/Finite failure category: Functional form of the failure intensity expressed in terms of time. Family/Infinite failure category: Functional form of the failure intensity function expressed in terms of the expected number of failures experienced. Type: The distribution of the number of the failures experienced by time \( t \). Poisson and Binomial are the two important types.

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

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

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

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

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

The Littlewood and Verrall model [1973] assumes exponential distribution for the random variable representing the failure interval time.
But the failure intensity is regarded as a stochastically decreasing function with gamma distribution, implying that the fault fixing process is not considered as perfect, and that faults are of different sizes. A user-controlled function determines the nature of the reliability growth. This model requires complex statistical inference for determining the parameters.

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

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

As we can see, the basic input to all these models is the times of past failures, or times between consecutive failures. These data are used to determine the value of parameters, and then to predict the reliability of the given software. Most of the models use calendar time, and where execution time is used, suitable methods are used to convert it to calendar time.
Goel and Okumoto proposed an imperfect debugging model called Goel and Okumoto Imperfect Debugging Model (Amrit and Goel [1985]), which assumes that faults are removed with certainty when detected, is not always the case. In this model, the number of faults in the system at time $t$, $X(t)$, is treated as a Markov process whose transition probabilities are governed by the probability of imperfect debugging. Times between the transitions of $X(t)$ are taken to be exponentially distributed with rates dependent on the current fault content of the system. The hazard function during the interval between the $(i-1)^{\text{st}}$ and $i^{\text{th}}$ failures is given by

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

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

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

$$f(t_i/\lambda_i) = \lambda_i e^{-\lambda_i t_i}.$$ And

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

where $\Psi(i)$ represents the quality of the programmer, and the difficulty of the programming task. It is claimed that the failure phenomena in
different environments can be explained by this model by taking different forms for the parameter $\Psi(i)$. This is a software reliability growth model based on stochastic differential equations for the integration testing phase of distributed development environment (http://www.coverity.com). This model has a simple structure, hence it is easily applied. This is very useful for software developers in distributed development environment in terms of practical reliability assessment.

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

The Schneidewind’s model is based on the fact that the current fault rate might be a better predictor of the future behavior than the observed rates in the distant past (Michael [1995]). The failure rate process may be changing over time and there are three forms of the model. Model 1: utilizes all of the fault counts from the n periods. This
reflects the view that all of the data points are of equal importance. Model 2: ignores the fault counts completely from the first through the s-1 time periods. ie. Use only the data from period s through n. This reflects the view that the early time periods contribute little if anything, in predicting future behavior. Model 3 is an approach intermediate between the first two, which reflects the belief that a combination of the first s-1 period is indicative of the failure rate process during the later stages.

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

Thomson and Chelson Model is a Bayesian Model category (Lakey et. al. [1997]). The hazard function for this model is defined as \((\bar{f}_i+f_0+1)/(T_i+T_0)\). Where, \(\bar{f}_i\) is the number of failures detected in each interval, and \(T_i\) is the length of testing time for each interval i.
Musa Execution Time Model assumes that there are N software faults at the start of testing, each is independent of others, and is equally likely to cause a failure during testing. A detected fault is removed with certainty in a negligible time, and no new faults are introduced during the debugging process. The process modeled is the number of failures in specified execution time intervals (Amrit and Goel [1985]). The failure rate, or the hazard function for this model is given by

\[ Z(\tau) = \phi f(N - nc) \]

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

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

\[ m(t) = N \left( 1 - e^{-\phi t} / (1 + \phi e^{-\phi t}) \right) \]

where,
N = total number of failures that would occur in infinite time
Φ = failure detection rate
φ = inflection parameter

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

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

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

\[
F(t) = 1 - e^{-\left(\frac{t}{\eta}\right)^\beta}
\]

Where \( F(t) \) = fraction of parts failing
\( t \) = failure time
\( \eta \) = characteristic life or scale parameter (MTTF)
\( \beta \) = slope or shape parameter
\( e \) = pi or 2.718281828

AMSAA Model is a NHPP model represented as: (Richard and Ray [2002]).
\[ \rho(t) = \lambda \beta t^{\beta-1}, t > 0, \lambda > 0, \beta > 0 \]

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

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

A hardware reliability growth model is used to mention product reliability in the period during which, the observed reliability advances towards the inherent reliability of the product. Hardware and software reliability predictions adjusted by their respective growth models to coincide with the same point in time can be combined to obtain a
prediction of the overall system reliability. There were a number of reliability growth models suggested for hardware reliability in the literature. In this section, we outline important models that discuss hardware reliability. Understanding the dynamic behavior of system reliability becomes an important issue in either scheduling the maintenance activities or dealing with the improvement in the revised system design. In doing so, the failure or hazard rate function should be addressed. Bathtub curve is usually adopted to represent the general trend of hazard rate function as shown in Fig. 2.3. This curve exhibits three distinct zones. The first is, the short initial period called variously the early failure, infant mortality, or the burn in period. The decreasing but greater failure rate early in the life of the system is due to one or more of several potential causes. The causes include inadequate testing or screening of components during selection or acceptance, damage to components during production, assembly, or testing, and choice of components which have too great a failure variability. It shall be a specific goal of the supplier to ensure that the early failure period is rigorously controlled and covered by a suitable warranty (Shooman [1968], Thomas [1973]).
The failures in the second zone are termed service failures. During this period, the failure or hazard rate is constant and it represents the effective life of the product.

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

Many studies were concentrated on depicting the geometric shape of the bathtub curve. The early contributors in this area include Bain [1974], Smith & Bain [1975], Gaver [1979], Hijroth [1980], Dhillon [1981], Lawless [1982], Jaisingh et. al. [1987], Haupt & Schabe [1992], Schabe [1994], Xie and Lai [1996], Edelstein [1998]. Wang et. al. [2002]) proposed a general form of bathtub shape hazard function in terms of reliability. The relation between hazard rate and reliability of a system follows the definition (Wang et. al. [2002]).
\[ Z(t) = -\frac{1}{R(t)} \frac{dR(t)}{dt} \]  

------------------------ (2.1)

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

\[ Z(t) = -\frac{1}{R(t)} \frac{1}{dt / dR(t)} = Z(R) \]  

------------------------ (2.2)

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

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

The data obtained from failure tests can be analyzed to obtain reliability, failure density, hazard rate and other necessary information (Srinath [1991]). Obviously, the behavioral characteristics exhibited by one class of components differ from those exhibited by another class of
components. In order to compare different behavioral characteristics and also to draw general conclusions from behavioral patterns of similar components, a mathematical model representing the failure characteristics of the components becomes necessary. The procedure involves assuming a function for hazard rate, and thereby obtaining reliability and failure density by using this failure rate function. The assumed function for the hazard rate will be the hazard model. Some of the common hazard models are discussed below:

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

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

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

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

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

The variation of failure rate, reliability, probability of failure, and failure density with respect to time for a constant hazard model is shown in the following figure Fig. 2.4 (Srinath [1991]).
Figure 2.4 Variation of Failure Rate, Reliability, Probability of Failure, and Failure Density for a Constant Hazard Model (Srinath [1991])

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

That is,

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

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

In the case of linearly increasing hazard model the hazard rate is assumed to increase linearly with time. That is, $Z(t) = Kt$, where $K$ is a constant.
That is, for a linearly increasing hazard model, Reliability, \( R(t) = e^{-\frac{Kt^2}{2}} \)

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

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

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

**Figure 2.5** Variation of Failure Rate, Reliability, Probability of Failure, and Failure Density for a Linearly Increasing Hazard Model (Srinath [1991])
It can be seen from the failure density curve that the curve has a slope equal to $K$ at time $t = 0$. Also the value of $f_d(t)$ reaches a maximum of $\sqrt{\frac{K}{e}}$ at time $t = \frac{1}{\sqrt{K}}$, and tends to zero as $t$ becomes larger.

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

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

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

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

The reliability can be expressed as

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

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

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

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

Following figure Fig. 2.6 shows the variation of reliability in case of Weibull model for various values of $K$ and $m$ (Srinath [1991]).
2.2.3 Reliability Models for Open Source Software

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

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

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

Martin et.al. [2005] have done exploratory interviews with free and open source developers to study the common quality practices among the developers to implement a quality process improvement strategy. They found that even though development of OSS projects share common practices the quality of the resulting products needs further empirical evaluation. This implies that we have to look into reliability models for open source software development.
An empirical study towards open source software reliability model was conducted by Ying and Joseph [2005]. They have collected data from eight active open source projects from “SourceForge.net” and reliability analysis was done based on the bug arrival rate. They claim that general Weibull distribution is a possible way to establish the reliability model. Further, in contrast with closed source projects, it is unlikely to find a Rayleigh curve, to model all open source projects.

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

2.3 Computational System Reliability

Computational system reliability is concerned with hardware reliability, software reliability, reliability of interaction between hardware and software and reliability of interaction between the system and the operator. In general, a system may be required to perform various functions, each of which may have a different reliability. In addition, at different times, the system may have a different probability of successfully
performing the required function under stated conditions. The analysis of the reliability of a system must be based on precisely defined concepts.

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

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

System-level reliability and availability requirements set forth by U.S. Government agencies procuring large software intensive systems encompass both hardware and software. However, specifications, statement of work requirements, and compliance documents (standards) usually implicitly or explicitly focus on hardware and are largely silent about software reliability, maintainability, availability and dependability.
Consequently, contractor system reliability analyses and design reviews usually ignore quantitative software reliability, maintainability, availability, and dependability requirements. During system testing and evaluation, data on software operating times, failure rates, and recovery times are not collected. Finally, logistics and support specialists devote significant attention to sparing and maintenance concept development, but often do not adequately consider the software-related sustainment issues of large computer systems. These problems can be solved, by an appropriate definition of requirements for software-intensive system reliability in specifications, and in the definition of programmatic requirements in contractual documentation (Myron et. al.[2007]).

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

2.4 A Framework to Enable the Early Prediction of Software Reliability

The objective is to develop a framework to enable the early prediction of software reliability incorporating reliability measurement in each stage of the software development. Leslie et.al.[2008] state that
the ability to predict the reliability of a software system early in its development can help to improve the systems quality in a cost effective manner. Therefore, the proposed framework, measures and minimizes the complexity of software design at the early stage of software development lifecycle, leading to a reliable end product. To calculate the reliability of software product, the reliabilities at different stages of product development like requirements analysis, design, development, testing and implementation etc. will have to be evaluated. This facilitates the improving of the overall product reliability. It is observed that modifications and error identifications during operation and implementation can lead to re-engineering of large parts of the system, which has been shown to be costly. Hence to ensure the quality of the developed system, it is important to ensure quality at different stages of development. A few approaches which do consider component-level reliability (Goseva et. al. [2003], Reussner et, al. [2003]) , assume that the reliabilities of a given component’s elements, such as its services, are known.

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

2.4.1 Background

The attention of scientists and engineers in the late 60’s was focused mainly on hardware reliability, mechanical and electronic
systems. Then from 70’s onwards the permanent growth of software applications became the center of many studies. Computers are applied in almost all areas of human life. The main applications includes banking system, power distribution, hospital management and critical systems like air traffic control and airplane flight, where failure could lead to catastrophes and loss of many lives (Vladimir et. al. [2011]). On one hand there is increasing dependence on software and on the other hand, software systems are becoming more and more complex and harder to develop and maintain. Software functionality is becoming crucial from the aspects of reliability, safety of human lives and security issues as well (Vladimir et. al. [2011], Voas and Payne [2000]).

2.4.2 Reliability Prediction

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

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

![Figure 2.7 Prediction Method](image)

### 2.4.3 The Framework

The main task of a system program office (SPO) when acquiring a new software system, is to specify the requirements to the developer and to see that the requirements mentioned are met as the system development process evolves to the final product. It is also necessary to assure that the qualities of the software such as reliability, maintainability, usability, testability, and portability are attained.
Figure 2.8 Detailed software reliability prediction framework
An error in the software product can occur when there is a difference between the actual output of the software and the expected correct output. Fault is a condition that causes a system to fail to perform its required function. Failure occurs when the behavior of the software is different from the specified behavior (Amitabha and Khan [2012]).

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

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

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

2.5 Techniques and Technologies for Measuring and Improving Software Reliability.

Reliability measurement is a set of mathematical techniques that can be used to estimate and predict the reliability behavior of software during its development and operation. The primary goal of software reliability modelling is to find out the probability that it will fail in a given time interval, or, what is the expected duration between successive failures (Allen and Lyu [1999], Allen and John [1998]). Software reliability is closely influenced by the creation, manifestation and impact of software faults. Consequently, software reliability can be improved by treating software faults properly, using techniques of fault tolerance, fault removal, and fault prediction. Fault tolerance techniques achieve the design for reliability, fault removal techniques achieve the testing for reliability, and fault prediction techniques achieve the evaluation for reliability (Lyu[1998]).
Reliability engineering is a daily practised technique in many engineering disciplines. Using a similar concept in these disciplines, we define software reliability engineering as the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability. Software reliability engineering therefore, includes (Lyu [1996]): (1) software reliability measurement, which includes estimation and prediction, with the help of software reliability models established in the literature; (2) the attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability; and (3) the application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance.

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

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