CHAPTER-I
INTRODUCTION
CHAPTER-I

INTRODUCTION

This chapter is introductory in nature. In section 1.1 we first briefly explain the significance of determining the reliability and availability of software system. Basic concepts of software reliability, its measures and its applications are presented in section 1.2. A brief survey of literature available on this subject is then discussed in section 1.3. In section 1.4 we explain in brief the important definitions and methods, on which our analysis is based on. Finally, we present in section 1.5, a brief summary of the work presented in the succeeding chapters of this thesis.

1.1 SIGNIFICANCE OF DETERMINING THE RELIABILITY AND AVAILABILITY OF THE SOFTWARE SYSTEMS

Software has made a crucial place for itself in our daily lives. Home appliances, mobile phones, automobiles, aircrafts, automatic teller machines, medical diagnostics and scientific applications are some of the examples where software plays a critical role in the functioning of the complete system. Earlier these systems were mainly comprised of hardware circuits, which required a complex procedure of replacement of hardware components, if some changes are to be made in the functionality of these systems. In order to reduce the size and complexity, in
many cases, complex hardware circuits have been replaced by software programs. With the
dawn of the internet and e-commerce major business and financial transactions are performed
on-line where software plays a crucial role. A small error in the software sub-system can cause a
failure in the complete system that leads to disastrous failures which differ in their impact
depending on the operations of an organization. A failure in the software sub-system used in
radiation therapy machine or in the aircraft control system can claim the precious human lives.
The financial system may collapse with an error in the software being used producing losses to
the business organizations. Therefore, the analysis of software systems for their reliability and
availability is of great significance in the present scenario.

The three most important characteristics of software product are quality, cost and
schedule. Quantitative measures exist for calculating cost and schedule but quantification of
quality has been more difficult. Reliability and availability are the most important measure for
evaluating the quality of the software system and represents user-oriented view of software
quality. It relates to operation rather than design of the program and hence it is dynamic rather
than static. Keeping this in view an attempt has been made in the present thesis to investigate
some aspects of the problems of reliability and availability of software systems, which still needs
further investigation.

1.2 SOFTWARE RELIABILITY MEASURES AND APPLICATIONS

The IEEE defines software reliability as the probability that software will not cause the
failure of a system for a specified time under specified conditions. There are three factors which
affects Software Reliability as defined by Musa et al. (1987):

(i) Fault Introduction: Faults are introduced when the code is being developed by the
programmers either during the original design or when new features are being added to the
software as per user requirements.
(ii) **Fault Removal:** Fault removal resulting from execution depends on the occurrence of the associated failure. Occurrence depends both on the length of time for which the software has been executing and on the execution environment.

(iii) **Environment:** Software reliability is directly dependent on the environment or the operational profile for the program, which is defined as the set of run types that a program can execute along with the probabilities with which they will occur. For example, a software system that is being used in a business organization will have software reliability different from that used by a user at home.

Software reliability measurement includes two types of activities as given by Musa et al. (1987): reliability estimation and reliability prediction. Reliability estimation determines current software reliability by applying statistical inference techniques to failure data obtained during system test or during system operation. This is a measure regarding the achieved reliability from the past until the current point. Wood (1996) defines reliability prediction as future software reliability based on available software metrics and measures. When failure data is available, the estimation techniques can be used to parameterize and verify software reliability models, which can perform future reliability prediction. When failure data are not available, the metrics obtained from the software development process and the characteristics of the resulting product can be used to determine reliability of the software upon testing or delivery. Software Reliability measures can be used in the following areas as defined by Musa et al. (1987):

(i) **Evaluation of Software Engineering Technologies:** New techniques are continuously being introduced for improving the process and reducing the cost and time of developing software. However, the new techniques are sometimes not evaluated quantitatively for their effectiveness. The software reliability and availability measures can be used to evaluate software engineering technology quantitatively and can be useful in comparison of the new process or technology with
the older one and can be helpful in taking decision about whether the new technology should be implemented or not.

(ii) Project Management during development: Software reliability and availability measure can be used for evaluating the development status during the test phase of the project. Methods such as perception of designers or test team, percent of tests completed and execution of critical function tests are generally used for evaluating the test phase of the project but these techniques have not produced satisfactory results. Reliability and availability benchmarks can be set at the start of the software development and testing should be done until the desired level of reliability is not achieved. Thus reliability measures can be used for evaluating the project schedules.

(iii) Operational Phase software management: The new features are added to software systems from time to time as per requirements of the user. Software reliability and availability measures can be used to monitor the operational performance of software, when changes are made to the software.

Lyu (1996) summarizes the following four technical areas which are applicable in achieving reliable software systems and they can also be regarded as four fault lifecycle techniques:

(a) **Fault prevention**: to avoid, by construction the fault occurrences in the software.
(b) **Fault removal**: to detect, by verification and validation, the existence of faults and eliminate them.
(c) **Fault tolerance**: to provide, by redundancy, service to the user with the specification in spite of faults having occurred.
(d) **Fault forecasting**: to estimate, by evaluation, the presence of faults and the occurrences and consequences of failures.
1.3 LITERATURE REVIEW

There are two commonly used approaches in the industry for software reliability modeling. The first approach utilizes software reliability growth models (SRGM) in later stages of software development cycle to estimate the software failure rate from the observed failures. As each failure occurrence initiates the removal of a fault, the number of failures that have been experienced by time \( t \), denoted by \( M(t) \), can be regarded as a reflected image of reliability growth. Each SRGM implicitly assumes a certain functional form of mean value function, \( M(t) \).

Using the failure times or the times between failures collected during a testing project, the parameters of a SRGM like expected number of failures by the time \( t \) or failure intensity can be estimated. Software architecture greatly affects the reliability and availability of software systems. The second approach referred to as architecture based modeling, utilizes discrete state Markov modeling in the software reliability estimation (Taylor and Vander (2007)). System failures can be avoided with better software architectures by raising exceptions when fault occurs and confining the failure to a particular module.

The first study of software reliability was done by Hudson in 1967. He described software development as birth and death process. The generation of fault is considered as birth and fault correction as death. The number of faults existing at any time defined the state of the process. He assumed that the rate of detection of faults is proportional to the number of faults remaining and rate of fault detection increase with time. Jelinski and Moranda (1972) developed software reliability growth model which is based on the assumptions that at the beginning of testing, there are \( u_a \) faults in the software code with \( u_a \) being an unknown but fixed number. Each fault is equally dangerous with respect to the probability of its instantaneously causing a failure. Furthermore, the hazard rate of each fault does not change over time, but remains constant. Whenever a failure occurs, the fault that causes it is removed instantaneously and
without introducing any new fault into the software. The Jelinski and Moranda model belongs to the binomial type of models as classified by Musa et al. (1987). The Jelinski and Moranda model has poor predictive capabilities in many cases. Another model proposed by Schick and Wolverton assumes that the hazard rate is proportional to the product of the number of faults remaining and the time. Schneidewind (1972) presented a software reliability model from an empirical viewpoint assuming fault detections per time interval as a non homogeneous Poisson process with an exponential mean value function. He applied least squares method or maximum likelihood estimation to the determination of the parameters of the process and also suggested that the time lag between failure detection and correction be determined from actual data and used to correct the time scale in forecasts.

Goel and Okumoto (1979) proposed a model assuming expected number of initial software faults as $N$, as compared to the fixed but unknown actual number of initial software faults $u_0$ in the Jelinski-Moranda model. Piwowarski et al. (1983) formulated a model for block coverage in dependence of the number of test cases executed during functional testing, which can easily be extended to become a model of the number of failures experienced in terms of time. Yamada et al. (1986) extended the Goel-Okumoto model stating that the ratio between the expected numbers of software failures occurring in $(t, t + \Delta t)$ with $\Delta t \rightarrow 0$ is proportional to the expected number of undetected faults. The Bayesian extension to the Jelinski and Moranda model was proposed by Littlewood et al. (1986) to improve the parameter estimation of the model and hence its predictive capability.

The Bayesian models can be viewed as a homogeneous Markov chain in a random environment. The model developed by Lyu (1996) was based on the assumptions that the number of failures experienced by time $t$ follows a Poisson distribution. The number of software failures that occur in $(t, t + \Delta t)$ with $\Delta t \rightarrow 0$ is proportional to the expected number of undetected
faults. Whenever a failure occurs, the fault that caused it is removed instantaneously and without introducing any new fault into the software. Since each fault is perfectly repaired after it has caused a failure, the number of inherent faults in the software at the beginning of testing is equal to the number of failures that will have occurred after an infinite amount of testing. Rivers and Vouk (1995, 1998) derived a hyper geometric model for the number of failures experienced at each stage of testing when the constructs tested are removed from the population. The constructs under study may be program size metrics like statements, blocks, paths, etc. They also considered the number of test cases executed and even execution time or calendar time. Gokhale et al. (1996) introduced the enhanced non-homogeneous Poisson process (ENHPP) model as a unifying framework for finite failure NHPP models with the assumptions that the N faults inherent in the software at the beginning of testing are uniformly distributed over the potential fault sites and at time t. Whenever a failure occurs, the fault that caused it is removed instantaneously and without introducing any new fault into the software. Huang et al. (2001) proposed the incorporation of testing effort into the modeling process, which can be measured by the human power, the number of test cases, or the execution-time information. Teng et al. (2006) studied reliability modeling of hardware and software interactions.

The above models follow black-box approach i.e., only the failure data from the software systems under measurement are included in the modeling process, while the system structures are ignored. Taylor and Hoek (2007) have described the importance of software architecture in calculating software reliability. Various studies for reliability and availability estimation have been carried out based on the architecture of software systems, (Pery and Wolf 1992). Tohma et al. (1989) gave a structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution. Tang and Iyer, (1992) presented analysis and modeling of correlated failures in multicomputer systems. Lee et al. (1993) presented measurement-based evaluation of operating system fault tolerance. Krishnamurthy and Mathur (1997) gave an
approach for the estimation of reliability of a software system using reliability of its components. Gokhale et al. (1998) have presented an analytical approach to architecture-based software reliability prediction. Smidts and Sova (1999) considered an architecture-oriented modeling approach for software reliability estimation based on decomposition of requirements into software functions and attributes. Kuball et al. (1999) introduced a hierarchical model to estimate the probability of failure on demand of a component-based software system.

Gokhale and Trivedi (1999) presented a software reliability modeling approach which included system structure as well as workload considerations. Wang et al. (1999) have also presented an architecture-based software reliability model. Goseva-Popstojanova et al. (2001) gave the comparison of architecture-based software reliability models. Chen et al. (2001) included testing coverage into time-basis adjustment for more accurate software reliability measurement. Boutremans et. al. (2002) has studied the impact of software failures on VoIP performance. Reussner et al. (2003), Roshandel and Medvidovic (2004) and Yacoub et al. (2004) have presented approaches for the reliability estimation of component-based software architectures. Cai and Lyu (2004) carried out an empirical study on reliability and fault correlation models for diverse software systems.

According to Lyu (2007) a recent trend in software architecture is that as information engineering is becoming the central focus for today’s businesses, service-oriented systems and the associated software engineering will be the standards for business development. Service orientation requires seamless integration of heterogeneous components and their interoperability for proper service creation and delivery. In a service oriented framework, new paradigms for system organizations and software architectures are needed for ensuring adequate decoupling of components, swift discovery of applications, and reliable delivery of services. Such emerging software architectures include cross-platform techniques (Bishop and Horspool, 2006), open-
world software (Baresi et al., 2006), and Web applications. Although some modeling approaches have been proposed to estimate the reliability for specific web systems (Wang and Tang 2003, Zhu et al. 2002, Suri and Bhushan 2007), software reliability engineering techniques for general web services and other service-oriented architectures require more research work.

System Availability is also a major performance concern especially in case of distributed systems. It represents the percentage of time the system is available to the users. Several studies have been done in this direction. Goel and Soenjoto (1985) and Sumita and Masuda (1986) were the first to propose a model for availability estimation. Balkovich et al. (1987) and Ibe (1988) have done availability analysis of VAX cluster system. Some models have been developed for the availability estimation of combined software hardware systems, such as the Stochastic Rendezvous Network Model. Hierarchical Model, presented by Hariri (1995), is a two-level model to analyze the availability of distributed systems. Lai (2002) presented a model for availability analysis for system with parallel redundant hosts and is a kind of 1-out-of-N- system. Mohanta (2005) proposed a fuzzy Markov model for determination of fuzzy state probabilities of generating units. Guillermo and Manic, (2006) have carried out fuzzy perform ability analysis of disk arrays. Suri (2009), has developed a simulator for risk assessment of software project based on performance measurement. Yadav and khan (2009) and Khaled (2009) have given a critical review of software reliability models.

In the analysis of certain software systems, few parameters such as software failure and repair rates cannot be exactly estimated. So, in the present scenario, it is difficult to analyze software reliability and availability due to such uncertain parameters. The models generally assume that once a fault is discovered it is removed immediately i.e. software’s have instantaneous repair time. The reality is that applications executing in the field can take significant amount of time may be days or weeks to get a fault removed. The second problem,
which is generally faced, is the quality of the failure data. For example repeat failures generally occur due to the fact that faults are not removed instantaneously. Another problem is that operational profile testing is generally ignored i.e. it is assumed that the software is going to be tested in the same manner that it is used in the field, which is not true in practice. Thus, there is a need of further improvements in the existing models so that reliability and availability can be computed more efficiently and accurately.

1.4 BASIC DEFINITIONS

1.4.1 Software Reliability

The IEEE defines software reliability as the probability that software will not cause the failure of a system for a specified time under specified conditions IEEE, (1988). It is the probability of failure-free operation of a computer program for a specified time in a specified environment. Software Reliability Engineering includes the following as defined by Verma et al. (2007):

(i) Software reliability measurement, which includes estimation and prediction with the help of software reliability models.

(ii) The attributes and metrics of product design, development process, system architecture, software operational environment and their implication on environment.

(iii) The application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use and maintenance.

1.4.2 Failure

A failure is said to occur in software when the actual output deviates from the expected output of the user. The program has to be executing for a failure to occur. Example of software failure can be the wrong result in a mathematical calculation performed in a program or a more
serious software failure may result in restart of the system software. Failure behavior is affected by two principal factors:

(i) The number of faults in the software being executed.

(ii) The execution environment or operational profile of execution.

The definition of failure varies from project to project and is established in consultation with the user. For example, a failure may be defined as a program crash requiring interruption of processing and program reload. A small variation in operation of the program can also be considered as failure for some other software.

1.4.3 Failure rate

Failure rate is expressed in terms of number of failures per unit time, e.g., failures per hour or failures per day or failures per year. It is denoted by \( \lambda \).

1.4.4 Repair Rate

Repair rate is expressed in terms of number of repairs per unit time, e.g., repairs per hour or repairs per day. It is denoted by \( \mu \).

1.4.5 Faults

A fault can be defined as a defect in software which causes a failure in it. It is also referred as a “bug”. Example of software fault is incorrect statement of a program. A single fault can cause multiple failures in software. A fault is a property of program rather than property of its execution or behavior. Faults are introduced during the program development or when new features are added to the existing programs.

1.4.6 Mean time before failures

Mean time before failures (MTBF) is the statistical average time before failures of the system.
1.4.7 *Software versus Hardware Reliability*

There are some fundamental differences between hardware and software failures that impact the analysis of hardware versus software reliability (Everett, 1999). The primary difference is in the underlying mechanisms causing failures. With hardware, the failures are generally caused by physical processes related to stresses imposed by the operating environment. Specifically, failures are due to components degrading, deteriorating, or being subjected to environmental shocks. With software, there is nothing to “wear out.” In a software system, failures never occur if the software is not used. This is not true of hardware systems where material deterioration can cause failures even though the system is not being used. Software reliability models are generally the analytical models derived from assumptions about the system, and the interpretation of those assumptions and model parameters. On the other hand, hardware reliability methods are usually derived from fitting specific distributions to failure data. This is done by extensive analysis as well as the domain experience. Finally, once defects in a software system are repaired, a new piece of software is obtained. This is not true of hardware repairs, which typically restore the original system. The primary mechanism for failures is the latent faults within the software. These faults may be the result of errors in coding or implementation of design or requirements specifications. However, just the presence of a fault is not enough to cause a failure. First the software must be executing, and second the input being processed during execution must be such that the fault will be encountered.

1.4.8 *Software Availability*

Availability is used to indicate the probability of a system or equipment being in operating condition at any time \( t \), given that it was in operating condition at \( t = 0 \).

\[
MTBF = \int_{0}^{\infty} R(t) \, dt \tag{1.1}
\]
1.4.9  Fuzzy Logic

Fuzzy logic is a logical system that generalizes classical two valued logic for reasoning under uncertainty. It refers to all of the theories and technologies that employ fuzzy sets which are classes with unsharp boundaries. Fuzzy logic has been viewed as a theory for dealing with uncertainty about complex systems. The distinguishing mark of fuzzy logic in rule-based systems is its ability to deal with situations in which making a sharp distinction between the boundaries of application in the use of rules or constraints is very difficult. The various applications of fuzzy logic includes consumer products, automotive and power generation, industrial process control, robotics and manufacturing and software reliability. Fuzzy logic offers an alternative to the probability paradigm, possibility, that is much more appropriate to software reliability when there is uncertainty in the data. Possibility mathematics allows for quantitative reliability calculations that preserve the uncertainty present in the original data, avoids making unwarranted assumptions and makes the consequences of the required assumptions clear throughout the analysis.

Fuzzy logic also improves reliability analysis through the concept of utility. Standard reliability models usually assume a binary representation of failure, a system is working or it is failed. A more flexible and realistic model allows for easy representation of partial system failures. Thus, there is considerable motivation to adapt the traditional probability based reliability methods to the fuzzy logic context.

1.4.10  Fuzzy Software Reliability

The various software reliability models contain parameters which are uncertain in nature. To incorporate the uncertainty these parameters are modeled as fuzzy numbers. Failure and repair rates are taken as fuzzy variables which are defined in terms of fuzzy numbers with suitable presumption level and membership grades. Triangular or trapezoidal possibility distributions can
be used in fuzzification process. The resulting fuzzified model offers more flexibility and accommodates uncertainties in failure and repair rates and other reliability measures. Fuzzified models offer several advantages to the software developer as they are more realistic and practical in nature than other models with rigid assumptions.

1.4.11 Fuzzy Set and Membership function

A set in a classical set theory has a sharp boundary because an element either completely belongs or does not belong to a set at all. Fuzzy set directly addresses this limitation by allowing membership in a set to be a matter of degree. The degree of membership in a set is represented by a number between 0 and 1; 0 means entirely not in the set, 1 means completely in the set and a number in between means partially in the set. A fuzzy set is defined by a function that maps objects in a domain of concern to their membership value in a set. Such function is called the membership function. The membership function of a fuzzy set \( A \) is denoted as \( \mu_A \) and membership value of \( x \) in \( A \) is denoted as \( \mu_A(x) \). The domain of membership function, is called the universe of discourse.

1.4.12 Basic Operations in Fuzzy Sets

(i) Fuzzy Disjunction Operation

A fuzzy disjunction operator is the maximum operator defined as

\[
\mu_{A \cup B}(x) = \max\{ \mu_A(x), \mu_B(x) \}
\]

(ii) Fuzzy Conjunction Operation

A fuzzy conjunction operator is the minimum operator defined as

\[
\mu_{A \cap B}(x) = \min\{ \mu_A(x), \mu_B(x) \}
\]
(iii) **Fuzzy Complement Operation**

The complement of a fuzzy set $A$ is defined by the difference between one and the membership degree in $A$.

$$\mu_{A^c}(x) = 1 - \mu_A(x)$$

1.4.13 **Fuzzy Numbers**

A fuzzy number is a number that is characterized by a possibility distribution or is fuzzy subset of real numbers (Guillermo and Manic, 2006). A fuzzy number is either a convex or a concave subset of the real line.

1.4.14 **Types of Membership Function**

A membership function can be designed by interviewing those who are familiar with underlying concept and later adjusting the values or by constructing automatically from data or feedback from the system performance. The most commonly used membership functions are parameterizable membership functions as their use reduces the system design time and it also facilitates the automated tuning of the system by making changes in the parameters.

(a) **Triangular membership Function**

A triangular membership function is specified by three parameters $[a, b, c]$ where $a$ is the lowest value, $b$ is the nominal value and $c$ is the maximum value.

$$\text{Triangle } (x : a, b, c) = \begin{cases} 
0 & x < a \\
(x - a)/(b - a) & a \leq x \leq b \\
(c - x)/(c - b) & b \leq x \leq c \\
0 & x > c
\end{cases} \quad (1.8)$$
A triangular fuzzy number is a fuzzy number whose membership function is of triangular shape. Triangular fuzzy number is used when fuzziness exists on both sides of the parameter.

Algebraic Operation of Triangular Fuzzy Number (T.F.N)

(i) The addition or subtraction of two T.F.N’s will also be T.F.N.
(ii) If operations such as multiplication, inverse and division are performed on two T.F.N’s, the resulting fuzzy numbers need not to be T.F.N.
(iii) If operations such as maximum and minimum are performed on two T.F.N’s, the resulting fuzzy numbers need not to be T.F.N.

Consider two T.F.N.’s A and B defined by Triplets such as A = [a, b, c] and B = [d, e, f], the various arithmetic operations on A and B are defined as follows

(a) Addition

\[ A (+) B = [a, b, c] + [d, e, f] \]

\[ = [a + d, b + e, c + f] \text{ is also a T.F.N.} \]

(b) Subtraction

\[ A (-) B = [a, b, c] - [d, e, f] \]

\[ = [a - d, b - e, c - f] \text{ is also a T.F.N.} \]

(c) Symmetry

\[-A= [-a, -b, -c] \text{ is also a T.F.N.} \]

(d) Multiplication, Inverse and Division

For other operations such as multiplication, inverse and division, T.F.N in the form of Triplet is not possible to be used.
(b) Trapezoidal membership Function

A trapezoidal membership function is specified by four parameters \([a, b, c, d]\) as follows:

\[
l(x : a, b, c, d) = \begin{cases} 
0, & x < a \\
(x - a) / (b - a), & a \leq x < b \\
1, & b \leq x < c \\
(d - x) / (d - c), & c \leq x < d \\
0, & x \geq d 
\end{cases}
\] (1.9)

A Trapezoidal fuzzy number (Tr. F. N) is a fuzzy number whose membership function is of trapezoidal shape. In this fuzzy number there is no single value, but we have a flat line over an interval \((b, c)\). These numbers are generally used when fuzziness exists on both sides of an interval.

Algebraic Operation of Trapezoidal Fuzzy Number

(a) If operations such as addition or subtraction are performed on two Tr.F.N’s, the resulting fuzzy number will also be Tr.F.N.

(b) If operations such as multiplication, inverse and division are performed on two Tr.F.N’s, the resulting fuzzy numbers need not to be Tr.F.N.

(c) If operations such as maximum and minimum are performed on two Tr.F.N’s, the resulting fuzzy numbers need not to be Tr.F.N.

Consider two Tr.F.N.’s A and B defined as \(A = [a, b, c, d]\) and \(B = [e, f, g, h]\), the various arithmetic operations on A and B are defined as follows

(i) Addition

\[
A (+) B = [a, b, c, d] + [e, f, g, h] = [a + e, b + f, c + g, d + h]
\]
is also a Tr.F.N.
(ii) Subtraction

\[ A (-) B = [a, b, c, d] - [e, f, g, h] \]

\[ = [a - e, b - f, c - g, d - h] \] is also a Tr.F.N.

(iii) Symmetry

\[ -A = [-a, -b, -c, -d] \] is also a Tr.F.N.

(iv) Multiplication, Inverse and Divisions

For other operations such as multiplication, inverse and division, Tr.F.N in the form of quadruplets is not possible to be used.

(c) Gaussian membership Function

A Gaussian membership function is specified by two parameters \{m, \sigma\} as follows:

\[
Gaussian \ (x : m, \sigma) = \exp \left( \frac{(x - m)^2}{\sigma^2} \right) \]  

(1.10)

where \( m \) and \( \sigma \) denote the center and width of the function, respectively. The shape of the function can be controlled by adjusting the parameter \( \sigma \). A small \( \sigma \) will generate a thin membership function while a big \( \sigma \) will lead to a flat membership function.

(d) Bell-shaped membership Function

A bell shaped membership function is characterized by three parameters \{a,b,c\} as follows:

\[
Bell \ (x : a, b, c) = \frac{1}{1 + \left| \frac{x - c}{a} \right|^{2b}} \]  

(1.11)
where the parameter $b$ is usually positive. This membership function is a direct generalization of the Cauchy distribution used in probability theory. A desired bell shaped membership function can be obtained by a proper selection of the parameters $a$, $b$ and $c$.

(e) $S$ membership Function

The $S$ membership function is a smooth membership function with two parameters: $a$ and $b$. The membership value is 0 for points below $a$, 1 for the points above $b$ and 0.5 for the midpoint between $a$ and $b$. The name of the function comes from the $S$ shape of the function.

\[
S(x : a, b) = \begin{cases} 
0 & x < a \\
\frac{2(x-a)^2}{(b-a)^2} & a \leq x < \frac{a+b}{2} \\
1 - \frac{2(x-b)^2}{(b-a)^2} & \frac{a+b}{2} \leq x < b \\
1 & x \geq b 
\end{cases} \tag{1.12}
\]

(f) $\Pi$ membership Function

$\Pi$ membership function is defined with two parameters and $b$. The function has a membership value 1 at point $a$, membership value 0.5 at $a-b$ and $a+b$, respectively. Unlike the $S$ function, the $\Pi$ function decreases towards zero asymptotically as we move away from point $a$.

\[
\Pi(x : a, b) = \frac{1}{1 + \left| \frac{x-a}{b} \right|^2} \tag{1.13}
\]
1.4.15 \( \alpha \)–Cuts

When we want to exhibit an element \( x \in X \) that belongs to a fuzzy set \( A \), its membership value should be greater than some threshold \( \alpha \in [0,1] \). The set of elements is the \( \alpha \)-cut of \( A \) denoted by \( A_\alpha \).

\[
A_\alpha = \{ x \in X, \mu_A(x) \geq \alpha \} \quad (1.14)
\]

Uncertain parameters can be modeled as Triangular fuzzy numbers or Trapezoidal fuzzy numbers using the following membership functions:

(i) Triangular Membership function: A triangular membership function is specified by three parameters \([a, b, c]\) as follows:

\[
\text{Triangle} \quad (x : a, b, c) = \begin{cases} 
0 & x < a \\
(x - a)/(b - a) & a \leq x \leq b \\
(c - x)/(c - b) & b \leq x \leq c \\
0 & x > c 
\end{cases}
\]

(ii) Trapezoidal Membership function: A trapezoidal membership function is specified by four parameters \([a, b, c, d]\) as follows:

\[
\text{Trapezoid} \quad l(x : a, b, c, d) = \begin{cases} 
0 & x < a \\
(x - a)/(b - a) & a \leq x < b \\
1 & b \leq x < c \\
(d - x)/(d - c) & c \leq x < d \\
0 & x \geq d 
\end{cases}
\]
The reliability and availability of the system is calculated for the following combinations of failure and repair rates for different $\alpha$-cut levels.

1.5 PRESENT WORK

In the present thesis we have studied the reliability and availability analysis of the following software systems:

(i) Web-based software system with three tier architecture
(ii) Application server software system
(iii) Router Software system
(iv) Clustered architecture system

The architecture of the systems and certain assumptions under which the system is working has been analyzed completely for finding their reliability and availability. In the present work we have also carried out fuzzy reliability and availability analysis for Application server software system, router software system and clustered architecture system, to incorporate such uncertain parameters. The fuzzy reliability and availability of the system has been calculated for the different combinations of failure and repair rates. The lower bound and upper bounds of fuzzy reliability and availability has been calculated for different $\alpha$-cut levels. In transient state the equations are solved numerically to calculate the reliability of the system and sensitivity analysis is carried out in transient state by studying the effect of software repair and failure rates of various subsystems on the reliability of the system. In steady state the set of kolomogrov differential equations are solved recursively as well as analytically to calculate the availability of the system and sensitivity analysis is carried in steady state also.

Chapter I is preliminary in nature that presents the fundamental definitions associated with reliability, availability and fuzzy logic. The membership functions used to calculate the parameters as fuzzy numbers have also been discussed. The chapter also presents a literature
review and general considerations of software reliability models followed by critical comparison and analysis of existing models. The methodology used in the thesis to carry out reliability and availability analysis has also been presented by taking a general four state Markov model.

Chapter II presents the reliability and availability analysis of Web-based software system with three-tier architecture. The analysis has been done by taking actual repair and failure rates of various components of the system. A Markov model for the system is developed and mathematical formulation has been carried out using mnemonic rule for sub-systems. The kolomogrov system of differential equations thus formed has been solved numerically to calculate the reliability and availability of the system. The sensitivity analysis has also been carried out in transient and steady state by studying the effect of failure and repair rates of various components on the reliability and availability of the system.

Chapter III presents the modeling and analysis of reliability and availability for application server using fuzzy logic. A mathematical model of Sun Java System Application Server system has been formulated using Markov method. The fuzzy reliability has been calculated for various combinations of failure and repair rates and fuzzy availability has also been calculated for different \( \alpha \)-cut levels. The sensitivity analysis of reliability to operating system recovery time has been presented by taking variations in the values of operating system recovery time. The sensitivity analysis of availability to application server instance failure rate and operating system failure rate has also been presented.

Chapter IV innovatively carries out the software reliability and availability analysis of router software system using fuzzy logic and Markov models. The uncertainty of the parameters has been easily incorporated in the fuzzy markov model by modeling the uncertain parameters as fuzzy numbers using triangular membership function. The behavior analysis of the system is carried out for various combinations of repair and failure rates of the sub-systems. The fuzzy
reliability and availability has been calculated for various combinations of failure and repair rates. The sensitivity analysis has finally been carried out in both transient and steady state by studying the effect of software failure and repair rates, on the reliability and availability of the system.

Chapter V highlights the fuzzy reliability and availability modeling of a system with clustered architecture that allows uncertainty based parameters of markov model in the mathematical reliability and availability evaluation. The fuzzy reliability and availability has been calculated for various combinations of failure and repair rates for different $\alpha$-cut levels. The sensitivity analysis has finally been carried out in both transient and steady state by studying the effect of software failure and repair rates, on the reliability and availability of the system.

Finally, Chapter VI summarizes the contribution of this thesis. The possible directions, along which further research work on the topic can be carried out, are also discussed in brief. The industrial significance and limitations of the present work are also presented in this chapter.