Chapter I
INTRODUCTION

Today modern computers are being widely used all over the world for solving the problems of science, engineering, business, education, communication etc. This use is based on their ability to operate at great speed, to produce accurate results, to store large quantities of information and to carry out long and complex sequences of operations without human intervention. Recently, it has been used for fast communication, as a source of information and for direct talk. The information processing is the fast growing industry in the world. Most of the customers of the information system are business firms.

A computer in the most general sense is a logical device whose operations are directed by sequence of instructions which is called a program. The basic hardware of the computer consists of logic and arithmetic circuits that perform various operations. The user of the computer interact with hardware as well as system software that makes easy to work with the computer. Software is an instrument / programme for transforming set of inputs into a discrete set of output and hardware is nothing but all the peripheral devices of the computer and the computer circuit itself.
INTRODUCTION

Today modern computers are being widely used all over the world for solving the problems of science, engineering, business, education, communication etc. This use is based on their ability to operate at great speed, to produce accurate results, to store large quantities of information and to carry out long and complex sequences of operations without human intervention. Recently, it has been used for fast communication, as a source of information and for direct talk. The information processing is the fast growing industry in the world. Most of the customers of the information system are business firms.

A computer in the most general sense is a logical device whose operations are directed by sequence of instructions which is called a program. The basic hardware of the computer consists of logic and arithmetic circuits that perform various operations. The user of the computer interact with hardware as well as system software that makes easy to work with the computer. Software is an instrument / programme for transforming set of inputs into a discrete set of output and hardware is nothing but all the peripheral devices of the computer and the computer circuit itself.
Software is the major part of most of the computer systems. There is a
great level of business competition among software producers. On the other
hand, software customers are most aware of the products and services available
to them. These customers, though dependent on their suppliers have become
increasingly sophisticated and demanding. Software producers understand the
needs of the customer, thoroughly and precisely. The most significant needs
are required level of quality, time of delivery and cost.

At the same time, development and operational cost of software have
increased. The size, complexity and degree of distribution of system have also
increased. Presently, many multicomputer systems are existing which are linked
by network. They may have many diverse pieces of software that run
simultaneously and interact with each other. On the other hand, development
cost are becoming higher and working with multicomputer system is extending
to a smaller organization as well as down to lower level of organization.

The operational effects of failure are large and often crucial due to big
size of software and its complicated structure. For example, effect of breakdown
of airline reservations, banking, automatic flight control, military defense and
safety control systems of nuclear power plant, etc.. The cost of failure include
not only direct expenses but also product's liability risk and damage to the
company's reputation. The later can have dramatic effect on company's share
market and profit.

Another factor is increasing pace of change in computing technology that is information systems become economically obsolete more rapidly. Software may have become outdated either as a result of new hardware technology or new software technology. Under these circumstances, it is becoming increasingly impossible to create a software product that is generous in the sense of providing simultaneously high quality, rapid delivery and low cost. The later two characteristics can be measured quantitatively but the quantification of the quality has been more difficult, because of the absence of a concrete measure of software quality.

In the rapid and competitive life, information processing is probably the most money affected industry in the world of economy. Today, and if any information or collected data needs processing the program should be ‘fault free’. Consequently, when many software systems are divided into segments that are developed by different companies, then a definite need exists to test their usefulness that is reliability.

Software means an instrument for transforming a set of inputs into a discrete set of outputs. It consists of set of coded statements whose functions may be to evaluate an expression and to store the results in a temporary or permanent location to decide which statement is to be executed next or to
perform input-output operations. Since, software are produced by a human being, the finished product is often 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. Here, the computing environment means, the physical machine, operating systems, compiler and translators utilities etc. These discrepancies are called as software faults. Basically, the software faults can be the results of ignorance of user requirement, ignorance of rules of computing environment and poor communication of software requirement between the user and the programmer. Even if we know that software contains faults, we generally do not know their exact identity and location. There are methods for indicating the existence of faults but these methods also has some limitations, because of the imperfectness of these approaches in assuring the correct programme. Thus, a measure is needed which gives the degree of programme correctness and can be used in planning as well as controlling additional resources needed for increasing the software quality.

'Reliability' is probably the most important of the characteristics inherent in the concept of software quality. It represents the user oriented view of software quality. Initial and many present approaches for measuring software quality were based on counting the faults or defects found in the programme. Usually either failures or repairs were counted and neither of them are equivalent
to faults. Even if, faults found are correctly counted, they are not a good status indicator. Whereas, faults remaining may indicate the status of a software.

Reliability is much richer measure. It is a customer or user oriented rather, it is related to operations and not with the design of programme and hence, it is dynamic rather than static. It is intimately connected with defects. Johnes (1986) has pointed out that defects represent the largest cost element in programming. Reliability relates directly to operational experience and the influence of faults on that experience. Hence, it is easily associated with costs. It is more suitable for examining the significance of trends, setting the objectives and predicting, when these objectives will be met.

1.1 Concept of Software Reliability:

There are many views regarding software reliability and method of quantifying it. Some people believe that this measure should be binary in nature so that an imperfect programme would have zero reliability while a perfect one would have a reliability value equal to one. Though, others feel that software reliability should be defined as relative frequency of the times that a programme works as intended by the user.

Here, the later point of view has been used, where the software reliability is a probabilistic measure. It is defined as, "the probability of failure free operations of a computer programme for a specified time in a specified
environment’. Shooman (1983) has described it as, ‘Software reliability is the probability that a given software system operates for specified time period without any software error, on the machine for which it has been designed, subject to the condition that it is used within design limits’. The concept of failure is widely used in the field of software reliability and it is defined as, ‘A failure is an unacceptable departure of programme operations from programme requirement’. Similarly, fault and error are also required to be defined to understand the software reliability. A ‘fault’ is the software defect that causes a failure and the programmer’s action that is, addition or omission that results in a fault is known as an ‘error’.

In the area of project management, software reliability provides a good measure for the evaluation of status and progress during the test phases of the project. Reliability generally increases by increasing the numbers of testing. If a reliability goal is set for a software project then it is possible to estimate, when that goal will be achieved during the system’s test period. Thus, software reliability theory provides a scheduling tool for managers. The time required to meet this goal is also the function of resources applied. This makes it possible to compute, test and debugging costs as a function of reliability.

A software reliability measure can be used to monitor software performance during the operational phase and to control design changes of the
programme. Design changes usually involve a decrease in reliability whereas reliability generally increases during a process of debugging. Software reliability offers a useful quantitative means of evaluating the effect of software engineering technology.

Hetch (1977) has defined three principal functions of software reliability namely measurement, estimation and prediction. Reliability measurement uses a failure interval data obtained by running a programme in its actual operating environment and reliability estimation uses failure interval data from a test environment. Estimation can be employed to determine the present or future reliability. Reliability prediction uses the programme characteristics (not the failure intervals) to determine the software reliability and most commonly takes into the account of factors such as programme size, complexity etc. Reliability is only function that can be performed during programme phases that is prior to the test.

Software reliability is closely linked with the size and complexity of software. The number of faults inherent in a programme and the mean time to failure of the programme at the start of the testing, are directly related to previous programme development process. Ottenstein (1979) has remarked that size and complexity are perhaps the main factors which influence reliability. In order to understand these relationships, one needs to understand the nature
of faults and how they are generated. On the contrary, software reliability measures will be useful tools for comparing and evaluating methodologies as well as pointing, where improvement is needed.

1.2 History of Software Reliability:

The first study of software reliability has been done by Hudson (1967) and he has viewed software development as a birth and death process. He has treated fault generation as a birth and fault correction as a death. The number of faults existing at any time defines the state of the process and transition probabilities are determined to describe fault correction. Hudson (1967), has obtained a Weibull Distribution of the time interval between failures. The rate of fault correction was assumed to be proportional to the number of fault remaining.

Jelinsky and Moranda (1972) has developed a model which assumes that hazard rate for failures is piecewise constant and proportional to the number of faults remaining. At each fault correction, the hazard rate is changed by a constant amount but remained unchanged between corrections. Maximum likelihood estimation has been used to determine the total number of faults existing in the software. Moranda (1975) has also proposed two variants of the initial model. In one variant, the hazard rate decreases at each correction but the step size forms a geometric progression. Shooman (1972) has presented a similar model.
He additionally postulated that hazard rate is proportional to fault density per instruction. He has also proposed several different fault correction profiles. More complex models of fault generation and correction process have also been proposed by Shooman and Natraj (1976).

Shick and Wolverton (1973) have proposed a somewhat different software reliability model. In this model, it is assumed that the hazard rate is proportional to the product of the number of remaining faults and the time spent for debugging. The debugging time between failures has a Rayleigh distribution. Schneidewind (1972) has taken an empirical approach to the software reliability modeling and emphasized the importance of establishing confidence intervals. He was the first person to suggest that time should be differentiated into operating time of the program and cumulative test time. Schneidewind (1975) in his later work, recommended that the time lag between failure detection and correction should be determined from the data and be used to correct the time scale in forecasts.

An important turn to this study has been given by Musa (1975). He has developed a theory based on earlier contributions but also introduced a number of new concepts. He has stated that software reliability theory should be based on execution time rather than calendar time. The execution time model is superior in simplicity, clarity of modeling, conceptual insight and predictive
validity. Musa model viewed execution time in two respects, the operating
time of a product and cumulative execution time that occurs during test phases.
The hazard rate has been assumed to be constant with respect to operating time,
but it varies as a function of remaining faults. Miyamoto (1975) has presented
evidences and verified the assumption that the hazard rate is proportional to the
number of remaining faults.

In a series of papers, Littlewood and Verall (1973, 1974 & 1978) have
applied a Bayesian approach to the software reliability theory. They have viewed
software reliability as a measure of strength of belief that a programme will
operate successfully in contrast to the classical approach. Whereas, in a classical
approach, reliability is being determined by an experiment that would count the
proportion of executions for which a programme operate successfully.

Several researchers have made different assumptions regarding hazard
rate in software reliability. Some of them assume that hazard rate is constant
with respect to programme operating time. However, other researchers postulate
that the value of hazard rate is a function of number of faults remaining.
Littlewood model assumes it as a random process in the term of failures
experienced. He proposes different functional forms to describe the variation
of the parameters in the random process with the number of failures
experienced. Littlewood (1980) has also proposed a differential fault model
which assumes that faults make different contributions to the programme failure rate, from being assessed with a different frequencies.

Goel and Okumoto (1978) have presented an imperfect debugging model that is intermediate in complexity between Musa and Littlewood model. This model views debugging as a Markov Process with various transition probabilities between states. An alternative model has also been formulated by Goel and Okumoto (1979). This model describes failure detection as a non-homogeneous poisson process. In this model, both the cumulative number of failures detected and the distribution of the number of remaining failure is found to be Poisson.

Keiller et al. (1983) have investigated a model similar to the Littlewood-Verali's general model. It characterizes the randomness of the hazard rates with the same distributions. But, it uses a different parameter of that distribution to express reliability change. For the study of basic concepts, underlying software reliability modeling and development of classification scheme, Musa and Okumoto have helped to clarify and organize, comparison as well as to suggest the possible new models [see Musa and Okomoto (1983)]. This work led to the development of Logarithmic Poisson execution time model [(see, Musa-Okomoto (1984)], which combines simplicity and high predictive validity. The Logarithmic Poisson model is based on a non-homogenous Poisson process with an intensity function that decreases exponentially with expected number
of failures. It is called Logarithmic because expected number of failures is a Logarithmic function of time.

Crow (1974) has proposed a model for reliability estimation of hardware systems during development of testing. It is non-homogenous poission process with a failure intensity function, that is a power function in time which can be applied to software with certain ranges of parameter value.

1.2.1 Software Reliability Models :

Three factors are very important in modeling of software reliability namely fault introduction, fault removal and the environment. Fault introduction depends primarily on the characteristics of the developed code and development process characteristics. The most significant code characteristic is size. The code can be developed to add features or remove faults. Fault removal depends on time, operational profile and the quality of repair activity. The environment directly depends on the operational profile. Since, some of the factors are probabilistic in nature, and operate over time, therefore software reliability models are formulated in terms of random process. The models are distinguished from each other in general by the probability distribution of the failure times or number of failures experienced or by the nature of variation of random process with time.
Software reliability models are based on a stable program execution in constant environment. This implies that neither the code nor the operation profile are changing. Thus the models focus mainly on fault removal. Most of the models take into account the effect of slow fault introduction. However, some models assume that the average or long term effect must be a decrease in failure intensity. If neither of the above three factors change then the failure intensity will be constant.

A good software reliability model enhances communication on a project and provides a common framework of understanding for the software development process. It also enhances visibility for the management and other interested parties. These advantages are valuable even if the projection made with the model in specific case are not particularly accurate. Software reliability models almost always assume that failures are independent of each other. The conclusion of independency is supported by a study of correlogram of failure data from 15 projects done by Musa (1979) and no significant correlation has been found.

Musa and Okumoto (1983) have developed a classification scheme for software reliability models. The scheme permits relationship to be derived for a group of models and highlights as well as suggests new models. It also reduces the task of model comparison,
Models are classified in terms of 5 different attributes.

1. **Time Domain**: Calendar time or execution time (CPU or processor).
2. **Category**: The number of failures which may be finite or infinite that can be experienced in infinite time.
3. **Type**: The distribution of number of failures experienced by time 't'.
4. **Class**: Functional form of the failure intensity in terms of time (for finite failure category only), and
5. **Family**: Functional form of the failure intensity in terms of the expected number of failures experienced (for infinite failure category only).

The classification approach have been chosen to be different for the two categories because of greater analytical simplicity and physical meaning. The models are differentiated basically in two categories depending upon the number of failures that can be experienced during infinite time is finite or infinite.

In case of infinite failure category models, the number of failures in infinite time is unbounded. In infinite category models, there are four types namely, $T_1$, $T_2$, $T_3$ and Poisson type models.

In case of finite failure category, two types namely Binomial type and Poisson type have been defined taking into account the distribution of failures.
experienced by time 't'. The model takes the name from the distribution of failures experienced by time 't'.

In Binomial type models there are following classes:

i) Binomial type exponential class [for detail study one can refer Jelinsky-Morando (1972) and Shooman (1972)]

ii) Binomial type Weibull class [see Shick-Wolverton (1973) and Wagner, (1973)]

iii) Binomial type C1 class [see Shieh-Wolverton, (1978)]

iv) Binomial type Pareto class [see, Littlewood, (1981)]

Whereas in Poisson type models, there are two classes:

i) Poisson type exponential class model [for details see Musa (1975), Schneidewind(1975), Moranda (1975), Goel and Okumoto (1979) etc].

ii) Poisson type Gamma class model [See Yamada et al. (1983)].

1.2.2 Poisson Type Model:

The Poisson process provides a good approximation to the occurrence of many real life events such as telephone calls, arrivals in a queue, insurance claims and so on. Under general Poisson type model, software failure process is studied using a non-homogenous Poisson process with failure intensity $\lambda(t)$. 
\[
P_{ij}(t, \Delta t) = \begin{cases} 
1 - \lambda(t) \Delta t + O(t) & ; \ j = i \\
\lambda(t) \Delta t & ; \ j = i + 1 \\
O(\Delta t) & ; \ \text{o.w.}
\end{cases} \quad *** \ (1.1)
\]

Let \( P_m(t) \) denotes the probability that \( M(t) \) is equal to \( m \) i.e.

\[
P_m(t) = P [ M(t) = m ] \quad *** \ (1.2)
\]

Thus,

\[
P_m(t + \Delta t) = \left[ 1 - \lambda(t) \Delta t \right] P_m(t) + \lambda(t) \Delta t P_{m-1}(t) + O(\Delta t),
\]

and the above equation can be solved for \( p_m(t) \) which is,

\[
P_m(t) = \frac{[\mu(t)]^m}{m!} \exp[-\mu(t)] \quad *** \ (1.3)
\]

where,

\[
\mu(t) = \int_0^t \lambda(x) \, dx \quad *** \ (1.4)
\]

And \( \lambda(x) \) is failure intensity function. The non-homogeneous Poisson process (NHPP) \( [ M(t), t \geq 0 ] \) has mean and variance equal to \( \mu(t) \), which is mean value function of the process.

Thus from equation (1.4), we can see that failure intensity is the derivative of mean value function and the process \( M(t) \) can be particularised by specifying
the mean value function or the failure intensity function.

Consider $T_i$ be a random variable representing the $i^{th}$ failure interval and $T_i$ is the random variable representing $i^{th}$ failure time. i.e.

$$T_i = \sum_{j=1}^{i} T_j^j; \quad i = 1, 2, \ldots$$

where $T_0 = 0$.

Consider the event $E_1$ that there are atleast ‘$i$’ failures experienced by time ‘$t$’ denoted by $[M(t) \geq i]$ and $E_2$ is the time to the $i^{th}$ failure and is at most denoted by $[T_i \leq t]$ then $E_1$ and $E_2$ are equivalent

$$[M(t) \geq i] \iff [T_i \leq t]$$

***(1.5)***

Using equation (1.3) and (1.5) the cumulative distribution function of $T_i$ can be obtained and is

$$P[T_i \leq t] = P[M(t) \geq i] = \sum_{j=1}^{\infty} \frac{[\mu(t)]^j}{j!} \exp[-\mu(t)]$$

Assuming that $m_e$ failures are observed during $(0, t_e]$, the poisson process $\{M(t), t \geq 0\}$ has independent increments and hence we obtain the conditional distribution of $M(t)$ given $M(t_e) = m_e$ for $t > t_e$ as
\[ P[M(t) = m | M(t_e) = m_e] = P[M(t) - M(t_e) = m - m_e] \]
\[ = \frac{[\mu(t) - \mu(t_e)]^{m-m_e}}{(m-m_e)!} \exp\{-[\mu(t) - \mu(t_e)]\} \]

** (1.6)

This is the distribution of additional failures during \((t_e, t)\) and from the equation (1.6) we can derive the conditional cumulative distribution function of \(T_i\) given \(M(t_e) = m_e\) where \(i \geq m_e\) as

\[ P[T_i \leq t | M(t_e) = m_e] = P[M(t) \geq i | M(t_e) = m_e] \]
\[ = \sum_{j=1}^{\infty} P[M(t) - M(t_e) = j - m_e] \]
\[ = \sum_{j=1}^{\infty} \frac{[\mu(t) - \mu(t_e)]^{j-m_e}}{(j-m_e)!} \exp\{-[\mu(t) - \mu(t_e)]\} \]

** (1.7)

The event \(T_{i-1} = t_{i-1}\) is equivalent to \(M(t_{i-1}) = i - 1\) for \(m_e = i - 1\) and equation (1.7) is the sum of Poisson probabilities except for one term. Hence,

\[ P[T_i \leq t_i | M(t_{i-1}) = i - 1] = 1 - \exp\{-[\mu(t) - \mu(t_{i-1})]\} \] ** (1.8)
\[ R(t_i' | t_{i-1}) = P[T_i' > t_i' | T_{i-1} = t_{i-1}] \]
\[ = 1 - P[T_i' \leq t_i' | M(t_{i-1}) = i - 1] \]
\[ = \exp[-\{\mu(t_i) - \mu(t_{i-1})\}] \quad ; \quad i = 1, 2, \ldots. \]

\[ *** \quad (1.9) \]

Recall that \( R(t) = 1 - F(t) \) and \( f(t) \) is derivative of \( F(t) \), hence taking negative of the derivative of equation (1.8) with respect to \( t_i' \), we get

\[ f(t_i' | t_{i-1}) = \lambda(t_{i-1} + t_i') \exp[-(\mu(t_{i-1} + t_i') - \mu(t_{i-1}))] \quad ; \quad i = 1, 2, \ldots. \]

\[ *** \quad (1.10) \]

The hazard rate is,

\[ Z(t) = \frac{f(t)}{R(t)} \]

and

\[ Z(t_i' | t_{i-1}) = \lambda(t_{i-1} + t_i') \quad ; \quad i = 1, 2, \ldots. \]

\[ *** \quad (1.11) \]

Program hazard rate for the Poisson type model is the same as the failure intensity function.
1.3 Concept of Bayesian Inference:

Before starting the main chapters, it is necessary to introduce some basic elements of Bayesian Theory such as loss function, prior distribution(s), posterior distribution, risk function and Bayes risk etc.

A short discussion on some of the elements is presented in the following subsections.

1.3.1 Loss Function:

An element of statistical decision theory, which is perhaps most important is the loss function and its specifications. Any decision making situation consists of a non-empty set \( \Omega \) of possible states of nature, also referred as the parameter space and a non empty set \( \mathcal{A} \) of actions available to decision maker. The nature chooses a point \( \theta \) in \( \Omega \) and the decision maker, without being informed of the choice of nature, chooses an action ‘a’ in \( \mathcal{A} \). As a consequence, there may incur some loss which will depend on ‘a’ and \( \theta \). Thus \textit{loss is a function of} \( \theta \) and ‘a’, say \( L(\theta,a) \). The function \( L(\cdot,\cdot) \) is known as Loss Function.

In point estimation problem, the action space consists of the set of all possible values of the parameter. Thus, it may be the whole parameter space or a subset of it. To decide an action a sampling experiment is conducted to collect the data. The data is considered to be an observation of a random variable \( X \)
which is assumed to have a probability distribution \( f(X | \theta) \), when the true state of nature is \( \theta \). Then the decision maker chooses an estimate or class of estimate \( \hat{\theta} \) as the value of the function of random variable \( X \), i.e. say \( T(X) \) for the given observed value \( X \). i.e. \( \hat{\theta} = T(X) \). The function \( T(.) \) is called the estimator and its value \( T(X) \) when \( X \) is observed is the estimate for \( \theta \). Naturally, the loss \( L(\theta, a) \), now reduces to \( L(\theta, T(X)) \) which is a random variate and depends on sample outcomes.

For point estimation a number of loss functions are available in statistical literature. These loss functions can be broadly classified into two groups, namely symmetric and asymmetric. Both types of loss functions have extensively been used in estimation problems, [see Berger (1985) and Martz and Waller (1982)]. The quadratic loss function is very popular in Bayesian Reliability analysis. The main reason behind its popularity is, that it was originally used in estimation problems when unbiased estimator of the parameters \( \theta \) were being considered. A second reason is the relationship with classical least square theory. Finally, the use of quadratic loss function makes the calculation relatively straightforward and simple. Moreover, a number of situations may arise in which squared error loss may be appropriate, especially when underestimation and overestimation are of equal importance.
1.3.2 Prior Distribution:

A prior distribution plays an important role in Bayesian analysis. This represents, whatever is known about the unknown parameters before the data is collected. Since knowledge about the parameters depend upon past experiences, it is difficult to choose a prior distribution that will be appropriate under all circumstances. Therefore, a broad class of prior distributions, which are likely to be useful in a variety of situations is often considered.

A general class of priors called non-informative prior refers to the case when a little or no information about parameter is known. A non-informative prior is proportional to square root of Fisher’s information, that is a prior distribution $g(\theta)$ of parameter $\theta$ can be defined as

$$g(\theta) \propto [I(\theta)]^{1/2} \quad \ast \ast \ast (1.12)$$

where $I(\theta)$ is Fisher’s information. This method is known as Jeffrey’s rule (see Jeffreys (1961)). The important feature of (1.12) is that, it is not affected by the restrictions on parameter space. There are other methods which has been suggested by various authors to find non-informative priors. For details, readers are referred to Box and Tiao (1973), Zellner (1971, 77), Villegas (1977), Berger (1985), Novick and Hall (1965), Akaike (1978), Bernardo (1979), Geisser (1984) etc..
In statistical literature, a number of informative priors have been suggested by authors such as g-prior by Zellner (1982), an independent t-gamma prior by Dickey (1975), Leamer (1978) and a conjugate prior by Raiffa and Schlaifer (1961) etc.

The non-informative and conjugate priors are commonly used in Bayesian analysis. However, any other weighting function can also be used as prior distribution. For detailed study about the choice of priors, one may refer to Berger (1985), Martz and Waller (1982), MacFarland (1971, 72), Martz and Lian (1977), Deelay et al. (1970), Jaynes (1968), Goel and De Groot (1981) etc..

1.3.3 Posterior Distribution:

The Bayesian method utilise both the sample information and the prior information about the parameters. Before collecting the sample data, the experimenter specifies a prior distribution say \( g(\theta) \), which has been discussed in the previous subsection. On the basis of the sample data, the experimenter specifies the likelihood function, say \( L(X|\theta) \). The prior information \( g(\theta) \) with sample information \( L(X|\theta) \), is then combined by Bayes theorem to get the posterior distribution \( \pi(\theta|X) \) as
\[ \pi(\theta \mid X) = \frac{L(X|\theta)g(\theta)}{\int_{\Omega} L(X|\theta)g(\theta) \, d\theta} \quad (1.13) \]

The posterior distribution \( \pi(\theta | X) \) is thus, an inferential statement in the Bayesian view.

1.3.4 Bayesian Point Estimation:

The Bayesian approach usually requires lesser sample data to achieve the same type of inference than those based on classical approach [see Mood et al. (1974)]. This approach is much more direct, that is deductive than classical approach which uses inductive reasoning. In fact the use of past experience makes Bayes inference more informative particularly in those situations where the prior distribution accurately reflects the variation of the parameter. In this sense the statistical inference based on classical approach seems to be more restrictive than Bayes inference. Also the classical approach becomes special case of Bayes approach for certain prior distributions. Bayesian Analysis provides better quality of inference, it reduces the sample size and thus time. Moreover, under natural identifiability and measurability conditions, Bayes estimators are consistent for almost all parameter values and very often enjoys small sample properties too. Perhaps, due to these reasons, this method is currently getting popularity in virtually all the areas of statistical applications.
[see for more details. Berger (1985)].

Now, consider that we wish to obtain a point estimate for \( \theta \) under some specified loss functions \( L(\theta, T(x)) \), where \( T(x) \) is the estimate of \( \theta \). In Bayesian approach, an estimate \( T(x) \) is selected such that it minimises the Bayes Risk. This is similar to the minimization of average loss for the specified posterior distribution. \( \pi(\theta | X) \).

A loss function which is often used for point estimation problem is squared error loss, given as,

\[
L(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2
\]

and may be considered as the error due to estimation. The Bayes estimator under the Loss function (1.14) [see Goon et al. (1980)] is the posterior mean and defined as,

\[
E(\theta | X) = \int_{\Omega} \theta \cdot \pi(\theta | X) \, d\theta
\]

1.4 Problem under investigation:

Computers are widely used all over the world for solving problems of science, engineering, medicines, business, education etc. In these applications, not only computer hardwares but software are also equally important. Growing applications of Computers, demand better quality of software. Reliability is
probably the most important characteristics inherent in the concept of software quality.

Investigations in reliability began in 1950's and the growth of the study started after 1960s. Later it was felt that the investigation is very useful for the study of performance of computers. The basis of the study was the assessment of hardware as well as software. A large number of research has been carried out in the field of hardware reliability whereas less attention has been given to the area of software assessment. Since, software are the major part of the most computer systems, the field of software assessment becomes a very important research area. Software reliability is concerned with better function of software to meet the requirement of the customer. The present research work is carried out in this area only. The total work is divided in five chapters.

The first chapter of this thesis is introductory. The aim of this chapter is to make the basis for chapters to be followed. At the beginning, the introductory chapter provides very general remarks about reliability as well as software reliability. The different models of software reliability and their classification criteria have also been given in brief. As per the classification criteria, there are two important categories of the models namely; Poisson Type Model and Binomial Type Model. The concepts which are needed for developing the theoretical part of this thesis, i.e. the Poisson type model, Bayesian Inference
and Prior distributions are discussed in detail. At the end of this chapter, the complete thesis has been summarised giving the details of each chapter.

The second chapter deals with estimation of the parameters of Poisson type exponential class model. At the beginning, the model is introduced along with the genesis of exponential distribution. The model, considered in this chapter, consists two parameters namely total number of failures and failure rate of a software (say, β0 and β1). The Bayes estimators of both the parameters have been obtained considering non-informative priors under squared error loss function. The obtained estimators have been compared with maximum likelihood estimators to assess the performance of proposed estimators on the basis of Monte-Carlo simulation technique after generating 1000 random samples.

In the third chapter, once again the estimation of unknown parameters of Poisson Type exponential class model is considered. Here, a more suitable prior distribution have been selected for both the parameters and the Bayes estimators of both the parameters have been obtained under squared error loss function for the selected priors. These obtained estimators have been compared with maximum likelihood estimator using Monte Carlo simulation technique after generating 1000 random samples.

The Weibull distribution plays an important role in reliability theory because of its great adaptability. Considering the usefulness of this model, a
Poisson type Weibull class model has been suggested for the use of software modeling by several authors. The problem of estimation of unknown parameters of this model has been considered in the fourth chapter. The Bayes estimators of unknown parameters have been obtained considering gamma prior and uniform prior. The obtained estimators are compared with maximum likelihood estimators on the basis of their efficiencies, applying monte-carlo simulation technique.

In the last chapter, a new class of model namely lognormal class has been established for the Poisson type. It seems, no attempt is made earlier to study this Poisson Type Lognormal class model in the context of software reliability. Here, first of all it has been proved that such model provides a good fit to the real data of software reliability. After verifying its usefulness on the basis of real data, maximum likelihood estimators for the parameters have been theoretically obtained and the same have been computed for the real as well as generated data.

At the end of the thesis a list of references has been provided.