CHAPTER 1

INTRODUCTION

Information processing is probably the most significant industry in the world economy today and in the foreseeable future. It has expand and continues to expand at a rapid rate. Since software is the major part of computer systems, the field of software engineering can expect similar rapid growth. Since there is more competition among software producers, software customers are more aware of products and services available to them. These customers once relatively naïve and dependent on their suppliers, have become increasingly sophisticated and demanding. Software producers must understand their needs thoroughly and precisely. Three of the most significant needs are level of quality required, time of delivery and cost. Reliability is probably the most important of the characteristics inherent in the concept Software Quality. Reliability is connected with defects and as we know defects represent the largest cost element in programming. Reliability assumes, totally or partially, many properties that are often quoted as aspects of quality. Reliability represents a user-oriented view of software quality. Initial approaches to measure software quality were based on attempting to count the faults or defects found in the program.

Reliability is a much wider measure. It is customer/user oriented rather than developer oriented. It relates to operation rather than design of the program hence it is dynamic rather than static. It takes account of the frequency with which problems occur. Further, it relates directly to
operational experience and the influence of faults on that experience. Thus reliability measures are much more useful than fault measures.

As computer applications became more diverse and spread through almost every area of everyday life, reliability has become very important characteristics of computer system. During the last three decades, the software reliability engineering has played a central role to provide several quantitative methods used in the real time software development processes. Since the assessment of software reliability is one of the main topics in this area, one needs any mathematical model to assess quantitatively the software reliability which is the probability that the software system does not fail during a specified time period. Measurement of software reliability involves estimation of software reliability or its alternate quantities from failure data. The term software reliability prediction is defined by Hechtr (1977), the process of computing software reliability parameters from program characteristic. Typically, software reliability prediction takes into account factors such as the size and complexity of a program, and it is normally performed during a program phase prior to test. The principal objective of a software reliability model is to forecast failure behavior that will be experienced when the program is operational. This expected behavior changes rapidly and can be tracked during the period in which the program is tested.

1.1 MOTIVATION

Software reliability is a very important and active research field in software engineering. There have been hundred of prediction models, since the first prediction model was published. Software reliability engineering typically focuses on directing test efforts after software has been created in order to achieve a predetermined failure intensity objective. Among all software reliability models, Software Reliability Growth Model (SRGM) is probably one of the most successful techniques in the literature, through hundreds of
publications. In practice, SRGM encounters major challenges. First of all, software testers seldom follow the operational profile to test a software, so what is observed during software testing may not be directly extensible for operational use. Secondly when the number of failures collected in projects is limited, it is hard to make statistically meaningful reliability predictions. Thirdly some of the assumption of SRGM is not realistic (Shooman 2002).

Although some historical SRGMs have been widely adopted to predict software reliability, researchers believe, they can further improve the prediction accuracy of these models by adding other important factors which affect the final software quality. As software complexity and quality are highly related to software reliability, the measurement of software complexity and quality attributes have been explored for early prediction of software reliability.

A normalized reliability growth curve is used to predict and verify whether software reliability is attained or not. It can be used to determine when to stop testing. The fact that there are so many software reliability models to choose from also intimates practitioners. So instead of investigating which models are suitable for their environments or which model selection criteria can be applied, researchers tend to simply take reliability measurements casually, and they are often suspicious about the reliability numbers obtained by the models of Deepak Pengoria and Saurabh Kumar (2009).

A software reliability model specifies the form of a random process that describes the behavior of software failures with respect to time. In this work we have taken a Non Homogenous Poisson software reliability model called Gompertz Model to predict the reliability of the software. It is shown that the proposed model can be derived from the well-known statistical theory of extreme value and has the quite similar asymptotic property to the classical Gompertz curve. We have applied the Gompertz software reliability model to assess the software reliability and to predict the number of initial fault contents.
As the parameters used in this model are unknown, to estimate these parameter there are two frequently used methods namely Least Mean Square Estimation (LMS) and Maximum Likelihood estimation (MLE). In this work we have practiced an alternative approach of Least Mean Square estimation to estimate the reliability model parameters. We empirically conclude that this new parameter estimation approach may function better than the existing estimation methods and is attractive in terms of goodness of fit test based on information criteria and mean squared error.

1.2 OBJECTIVE OF THE WORK

The principal objective of this work is to forecast failure behavior that will be experienced when the program is operational. Measurement of software reliability involves estimation of software reliability or its alternate quantities from failure data. There are two alternative ways of expressing software reliability, the failure intensity is, the expected number of failures per unit time and the mean time to failure (MTTF), if it exists, is defined as the expected value of the failure interval. Even a software reliability growth model that yields accurate parameter estimates in the early testing phase is not practical if software engineers and managers cannot decide on the most appropriate model for their application. The criterion must be able to identify the most appropriate model in the early testing phase. A model that fits the data well in the early testing phase can provide a good fit in the later phase. However, software engineers and managers have two problems. One is how to get parameter estimates of software reliability growth models. The other is that engineers and managers have had little guidance as to which models may be the best for a particular application (Daisuke Satoh and Shigeru Yamada, 2001).

In this work, the Gompertz reliability model (S- Shaped growth model) is considered, because recently discrete analogs of software reliability model has been used for many web applications. Satoh (200) described a
software reliability growth model that yields accurate parameter estimates even with the small amount of input data. The model is based on a proposed discrete analog of a Gompertz equation that has an exact solution. The discrete equation that has an exact solution is easily applied to a regression equation to get parameter estimates and has advantages compared to an ordinary difference equation. It yields accurate parameter estimates in the early testing phase (Satoh 2001). The use of Gompertz model for reliability growth is restricted to data with estimates or predictions of reliability values, at the onset of growth, which are less than 50% (Dimitri Kececioglu et al 1994). From several data sets, it was seen that in general, reliability growth data with an S-shaped trend could not be described well by the conventional Gompertz model, because this model has a fixed value of reliability at its inflection points. Therefore, the parameter estimation approaches used in this model is modified in this proposed work. The resulting approach may be more flexible than its predecessor.

Many existing software reliability models are based on the assumption of statistical independence among successive software failures. However, some important issues that are crucial to the proposed modeling framework to be used in practice remain untouched such as the method of estimation of model parameter. In this work, the issue of parameter estimation problem for the software reliability modeling framework is addressed. A relationship function among model parameters and the reliability prediction using the Gompertz model framework is proposed. A three parameter estimation method based on different types of data available using Least Mean Square (LMS) estimation is proposed. In general the reliability at time ‘t’ is estimated using two input values namely time interval and number of failures. The reliability estimation formula used in all the models is $R = \exp(-\lambda t)$, where $\lambda$ is number of failures at time ‘t’. In Gompertz model the reliability $R$ is calculated using the above formula, but in the real time project testing phases,
test cases are defined for each module and accordingly the testing processes are carried out. The testing result also based on number of test cases written for a particular module. Hence in our proposed work, while estimating the parameters for Gompertz model, number of test cases is considered as one of the parameter. So a correlation between number of failures and number of test cases is established. More importantly this relationship can be used to obtain other model parameters that are needed to conduct software reliability prediction. The three parameter estimation method is proposed based on the collected data. This method is been practiced on real time project and observed the reliability prediction accuracy. The proposed parameter estimation method is implemented in web based applications.

1.3 SOFTWARE RELIABILITY ENGINEERING (SRE)

Software testing is an important process of software development to ensure the quality of software products. For large and complex system, testing becomes more complex. Although much research has advanced the techniques for generating test cases with high defect coverage, testing to guarantee defect free software remains difficult. Alternatively an empirical software reliability prediction technique is used to estimate the reliability of the software product (Rattikorn Hewetti et al 2006). The software reliability prediction is defined as the forecast of, how reliable an executable software system will be at some point in the future based on data available now. This is based on the industry definition of software reliability namely the probability of failure-free execution of a software system for a specified time in a specified operating environment. There is a difference between estimation and prediction of software reliability. The estimation is an assessment of how reliable a software system is now based on observed test data. Prediction is usually limited to a project period prior to system test. In other words, for prediction a development organization takes information about the system under development and uses
some statistical regression model to forecast the level of reliability that will be present at some point in testing (Peter Lakey 2002).

Reliability is probably the most important characteristics inherent from software quality. It is intimately connected with defects, it is the probability that the software without failure for a specified period of time. In addition to its preeminent importance, software reliability has proved to be the most readily quantifiable of the attributes of software reliability. Reliability is a much richer measure. It is a customer or user oriented rather than developer-oriented. Thus reliability measures are more useful than fault measures (John Musa et al 1987). As faults are removed, as in test phase, failure intensity tends to decrease and reliability increases. When faults are introduced during operation or test, as in cases when new features or design changes are being introduced into the system or when faults predominate repairs during debugging, there tends to be a step increasing in failure intensity and a step decreasing in reliability. If a system is stable, as in a program that has been released and there is no changes in code, both failure intensity and reliability tend to be constant (Dong Nguyen and Thomoson 2001). In reality, SRE tasks are fundamentally linked to both software and test engineering. SRE is just a quantitative perspective of software quality management (Koji Ohishi et al 2005 and Amrit Goel 1985).

As reliability is concerned with frequency of different types of failures we need to have a clear and unambiguous classification of failures. The failure classification scheme should be general and comprehensive and should permit a unique classification of each failure. This failure classification will have to be from the user’s perspective. For a modern software product, the failures can be partitioned at the top level as unplanned events, planned events and configuration failures. Unplanned events are traditional failures like crash, hang, incorrect or no output, which are caused by software bugs. Planned
events are those where the software is shutdown in a planned manner to perform some task. Configuration failure occurs due to problems in configuration setting. Different products may choose to focus on specific types of failures only, depending on what is of importance to their users and the overhead of measurement. It can be argued that planned events and configuration failures are not software failures, as there is no software fault causing them, but as they affect the user’s reliability experience.

Reliability quantities have usually been defined with respect to time, although it would be possible to define them with respect to other variables. Time may be considered in three ways:

- Execution time (CPU time)
- Calendar time
- Clock time

There are four general ways of characterizing failure occurrences in time:

- Time of failure
- Time interval between failure
- Cumulative failures experience up to a given time
- Failures experienced in time interval

The above said processes are random because the error occurrence in a program is random. The location of error occurrence is also random. There are two main reasons for this randomness. First, the commission of errors by programmers, it introduces fault, and it is very complex and unpredictable process. Second, the condition of execution of program is generally unpredictable. The fault behavior is affected by two principle factors:

- The number of faults in the software being executed, and
- The execution environment or operational profile of execution

Faults are introduced when the code is being developed by programmers. They may introduce the faults during original design or when they are adding new features, faults that have been identified. Fault removal is dependent on the efficiency with which faults are found and removed. Thus the failure process and the software reliability is directly dependent on the environment or the operational profile for the program (Rattikorn Hewetti et al 2006).

Assuring software to high levels of reliability is one of the most difficult yet important challenges confronting the software industries. Mathematical verification techniques may eventually become available for large control program, but they cannot address the problem of faults in the specification. Most of the research addressing ultra-high reliability involves one of three techniques; design diversity, testability and program self checking (Norman Fenton and Shari Lawrence Ptheger 2002).

The study of reliability as the quantification of the operational behavior of software systems with respect to user requirements is defined as software reliability engineering. The classical SRE process follows the four steps;

i. **Reliability objective:** It relates to what the reliability goal of the software is from the view point of the customer. This reliability objective is related to what kind of system failure the user wants to measure. For each class of failure identified, the reliability requirement can then be defined using an appropriate reliability metric.

ii. **Operational behavior:** It relates to the information obtained through the system operation on a certain environment. The
construction of an operational profile is important in order to select test cases according to the usage of the system. A fault must be executed to cause a failure, otherwise it is just a dormant fault. Also the importance of the operational profile is endorsed by the fact that software reliability is tightly related to the environment where the software is being executed.

iii. **Reliability modeling:** It is essential to the reliability prediction and estimation process. Most of the reliability modeling approaches attempt to predict software reliability in the latest stages of the life cycle. The software reliability estimation determines if a product meets its reliability objective and is ready for release. To do this, the failure data should be collected and apply it on a model. Using some statistical methods, the best estimates of reliability are obtained. If the reliability objective is not met, more testing will take place in an iterative manner.

iv. **Reliability validation:** It consists of comparing the projected reliability with the observed reliability. This validation will then provide a confident level of the reliability estimates as well as feedbacks to the reliability engineering process and software enhancement.

Figure 1.1 shows an overview of the framework for the reliability assessment of software components, which combines statistical testing. It shows the key components in SRE process including Reliability objective specification, operational profile determination, reliability modeling and measurement and reliability validation.
Figure 1.1 Software Reliability Engineering Process overview

To enhance a company’s ability to deliver timely, high quality products through an application of SRE practices several elements to be investigated. First, customer usage is quantified by developing an operational profile. Second quality is defined quantitatively from the customer’s viewpoint by defining failures and failure severities by determining a reliability objective. Then advocate the operational profile to manage resources and to guide design implementation and testing of software.

A reliability objective is the specification of the reliability goal of a product from the viewpoint of the customer. Operational profile is a set of disjoint alternatives of system operation and their associated probabilities of occurrence. The construction of an operational profile encourages tester to
select test cases according to the system’s operation usage, which contribute to more accurate estimation of software reliability in the field. Several interdependent estimates equivalent statements about software product’s reliability, they includes, failure intensity as a function of test time, the number of failures expected up to test time and mean time to failure at test time. These reliability estimates can provide the following information;

- The reliability of the product at the end of system testing.
- The amount of test time required to reach the product’s reliability objective.
- The reliability growth as a result of testing.
- Finally, the projected field reliability has to be validated by comparing it with the observed field reliability.

Software reliability requirements are specified during earlier development phases and SRE techniques are used to estimate the resources that will be required to achieve those requirements during test and operations. Reliability and its requirements can be expressed in one of the three following ways;

- Probability of failure-free operation over a specified time interval
- MTTF
- Expected number of failures per unit time interval.

Reliability and reliability related requirements must be stated in quantitative terms, otherwise it will not be possible to determine whether the requirements have been met or not. The SRE process involves steps such as defining necessary reliability, developing operational profile, preparing for test,
executing text and applying test data to guide decision related to release of the developed software (Behrouz Farl 2007).

As software reliability represents a customer-oriented view of software quality, it relates to practical operations instead of the design of programs merely. Therefore, it is dynamic rather than static. The aim and objective of software reliability engineering are to increase the probability that a completed program will work as intended by the customers. Hence measuring and computing the reliability of a software system are very important (Cin-Yu Huang and Michael Lyu 2003). Software Reliability engineering is an established discipline that can help organizations improve the reliability of their products and processes. The IEEE defines SRE as “the application of statistical techniques to data collected during system development and operation to specify, predict, estimate and access the reliability of the software – based systems”, (Norman Schneidewindo 2009).

As reliability is concerned with frequency of different types of failures, there is a need to have a clear and unambiguous classification of failures. The failure classification scheme should be general and comprehensive and should permit a unique classification of each failure. This failure classification will have to be from the users’ perspective. For a modern software product, the failures can be partitioned at the top level as unplanned events, planned events and configuration failures. Unplanned events are traditional failures like crash, hang, incorrect or no output, which are caused by software bugs. Planned events are those where the software is shutdown in a planned manner to perform some task. Configuration failure occurs due to problems in configuration setting. Different products may choose to focus on specific types of failures only, depending on what is of importance to their users and the overhead of measurement. It can be argued that planned events and configuration failures are not software failures as there is no software fault causing them, but as they affect the users’ reliability experience.
Software reliability can also be predicted using benchmark measurement and software process data before software test (Hong Li Wu et al 2010). Benchmarking is the process of comparing the business processes and performance metrics including cost, cycle time, productivity or quality to another that is widely considered to be an industry standard benchmark or best practice. Benchmarking provides a snapshot of the performance of measure process and helps us to understand where we are in relation to particular standard.

System test and field trial activities certify that the requirements of the product are met and that the product is ready for general use by the customer. The system test stage is critical, since it is the last stage in the development process where corrective action can be taken to improve the reliability of the product before release to the first customer. In reliability growth testing, system testers execute test cases in proportion to how often their corresponding operations occur in the field, as characterized by the operational profile. By mirroring customer use, reliability growth testing is likely to find the failures that causes the greatest customer dissatisfaction and will reflect the reliability the customer will experience. The goal of reliability growth testing is to attain a level of confidence that the software product is being released with reliability that meets customers’ needs. It is the last step in system test.

The practice of SRE continues during post delivery and maintenance. Field reliability is monitored against established product objectives and customer satisfaction. This can be used to improve the reliability of future product release and to improve the quality of the development process.

1.4 SOFTWARE RELIABILITY MODELS (SRM)

In the field of software reliability engineering, one particular aspect that has received the most attention is software reliability modeling. It is
rational, since all activities in SRE are based on models established in literature. Models are the basis of SRE. In the past 3 decades, research activities in software reliability engineering have been studied and more than 50 models have been developed. In this work a Non Homongeneous poisson Process (NHPP) Gompertz model is considered. It is an S- shaped curve model and it is suitable for web based applications. A software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect, fault introduction, fault removal and the operational environment (Michael Lyu and Allen Nikoral 1992).

From Figure 1.2 it is seen that the failure rate is decreasing due to the discovery and removal of software failures. At any present time, it is possible to observe a history of the failure rate of the software. Software reliability modeling forecasts the curve of the failure rate by statistical evidence. Thus the testing time and the reliability of the software could be predicted.

![Software Reliability Growth](image)

**Figure 1.2 Software Reliability Growth**

Software reliability growth models are designed to make predictions. Predictions of actual reliability or failure rate time needed to reach a given reliability target and things like that. In practice, software reliability growth
models encounter major challenges. First of all, software testers seldom follow the operational profile to test the software, so what is observed during software testing may not be directly extensible for operational use. Secondly, when the number of failures collected in a project is limited, it is hard to make statistically meaningful reliability predictions. Thirdly, some of the assumptions of Software reliability Growth Model (SRGM) are not realistic e.g. the assumption that the faults are independent of each other, that each fault has the same chance to be detected in one class; and that corrections of a fault never introduces new fault (Deepak Pengoria and Saurabh 2009).

Vladimir Zeljkovic et al (2011), the authors have made a study on reliability and show that the software reliability cannot be calculated during the design phase. If adequate data on system failure is collected throughout the project during testing phase, the models could apply on the parameter to predict the reliability. They have also observed that the importance of reliability estimation during the testing phase.

Most reliability growth models depend on one key assumption about software system-faults are identified there by increasing the reliability of the software. The data on failures and fixes for these models is typically obtained during the final stages of testing. The growth model is used to predict the reliability of the software system at any point in time during this failure and fix process. The key issue is to obtain a good model that can explain the past data and predict the future Pankaj Jalote et al (2004).

The SRGM falls into two categories; Time between Failure model (TBF) which treats the inter-failure interval as a random variable and Failure Count (FC) models, which treat number of failures in a given period as a random variable. In case of TBF models the parameters of inter-failure distribution change as testing proceeds, while the software reliability evolution in FC models is described by letting the parameters of distribution such as mean
value function, be suitable functions of time. One of the basic assumptions common to both classes of models is that the failures, when the faults are detected are independent (Katerina Goseva 2000). The basic principle of time domain software reliability modeling is to perform curve fitting of observed time-based failure data by a pre-specified model formula, such model can be parameterized with statistical technique. The model can then provide estimation of existing reliability or prediction of failure reliability by extrapolation techniques. Estimating remaining defects in software can help test managers make release decisions during testing. Several methods exist to estimate defect content, among them a variety of software reliability models. Software reliability growth model have underlying assumptions that are often violated in practice, but empirical evidence has shown that many are quite robust despite these assumption violations. The problem is that because of assumption violations, it is often difficult to know which models to apply in practice (Stringfellow and Amschler Andrews 2002 and Bev Littlewood 1979).

Software reliability models usually make a number of common assumptions, such as;

- The operation environment where the reliability is to be measured is same as the testing environment in which the reliability model has been parameterized.
- Once a failure occurs, the fault which causes the failure is immediately removed.
- The fault removal process will not introduce new faults.
- The number of faults inherent in the software and the way these fault manifest them to cause failures follow at least in a statistical sense, certain mathematical formulae.
Reliability modeling is an essential element of the reliability estimation process. It determines whether a product meets its reliability objective and is ready for release. One or more reliability models are employed to calculate, from failure data collected during system testing, various estimates of a product’s reliability as a function of test time. Several interdependent estimates can be obtained to make equivalent statement about a product’s reliability. These reliability estimates can provide the following information:

- The reliability of the product at the end of system testing
- The amount of test time required to reach the product’s reliability objective
- The reliability growth as a result of testing
- The predicted reliability belong the system testing (Michael Lyu 2007).

These models have been useful to the extent that many different approaches have been explored. However, software engineers and managers have two problems. One is how to get parameter estimates of software reliability growth models. The other is that engineers and managers have had little guidance as to which models may be the best or which may be the best for a particular application (Daisuke Satoh and Shigeru Yamada 2001).

1.4.1 Classification of Software Reliability Models

Both static and dynamic software reliability models exist to assess the quality aspect of software. These models aid in software release decisions. A static model uses software metric to estimate the number of defects in the software. A dynamic model uses the past failure discovery rate during software execution or cumulative failure profile overtime to estimate the number of failures. It includes a time component, typically time between failures.
Dynamic models measure and model the failure process itself and because of this, they include a time component, which is typically based on recording time $t_i$ of successive failure $i$ ($i \geq 1$). Time may be recorded as execution time or calendar time. These models focus on the failure history of software. A general assumption of these models is that software must be executed according to its operational profile. Most of the software growth models works under assumption that reliability of software grows due to the bugs that cause failures being removed from the software. While correcting the bugs, it will improve the reliability. Another assumption is that the failure rate of a software product improves with time irrespective of whether bugs are corrected or not. The reliability of software products depends on the length of time the users have been using the products. One of the reasons for this reliability growth is that as the users gain experience with the product, they learn to use the product correctly and find work around for failure causing situation (Pankoj Jalote and Brendan Murphy 2008).

Models differ based on their assumptions about the software and its execution environment. Some of the well known models are, the basic Musa model, the delayed S-shaped model, the Yamada exponential model etc. These models represent a range of assumption. The Musa model, the Goel-okumoto model, the Gompertz model and the Yamada exponential model assume testing following an operational profile. They also assume that the software does not change, except that defects are fixed when discovered. The models differ in their assumptions either in terms of workload, error size or failure density.

There are many ways to develop a software reliability model. They are describing as a stochastic process, relate it to a Markov model, define the probability density or distribution function and specify the hazard function. There are three classes of software prediction model. They are exponential
(NHPP), Non-Exponential NHPP models and Bayesian models (Katerina Gosva et al 2000).

i. **Exponential NHPP Models**

Exponential NHPP models use the stochastic process and the hazard function approach. The hazard function, \( Z(t) \), is generally a function of the operational time, ‘t’. The probability of success as a function of time is the reliability function, \( R(t) \), which is given as follows;

\[
R(t) = e^{-\int \! Z(t) \, dt}
\]

Sometimes reliability is expressed in terms of a single parameter, mean time-to failure (MTTF). MTTF is given as;

\[
MTTF = \int_{0}^{\infty} R(t) \, dt
\]

The generalized exponential model contains the ideas of several well known software reliability model. The main idea is that the failure occurrence rate is proportional to the number of faults remaining in the software. Further, the failure rate remains constant between failure detection and the rate is reduced by the same amount after each fault is removed from the software. The main objective of this model is to predict the following:

- Number of failures that will occur by a given time.
- Maximum number of failures that will occur after a given time.
- Number of fault corrected by a given time.
- Reliability model for the software after release.

The assumptions of the generalized exponential model are the following:
- The failures are likely to occur and are independent of each other.
- Each failure is of the same severity as any other failure.
- The faults that caused the failure are corrected immediately without introducing new fault into the program.

Representative model in this class include Scheidewind’s model, Sahoo’s model, Musa’s basic model, Jelinski and Moranda’s model. The generalized exponential model has the following limitations:

- It does not account for the possibility that each failure may be dependent on others.
- It assumes no new faults are introduced.
- It does not account for the possibility that failures can be increased overtime as the result of program changes, although techniques for handling this limitation have been developed.

The major model applications are:

- **Predicting**: Predicting the future failures and fault correction
- **Control**: Comparing prediction results with predefined goals and flagging software that fails to meet those goals.
- **Assessment**: Determining what action to take for software that fails to meet goals.

**ii. Non-Exponential NHPP models**

Non exponential NHPP models also use the stochastic process and the hazard function approach. They are applicable after completion of testing. Early fault corrections have a larger effect on the failure intensity function than
later ones. The logarithmic Poisson model is applicable when the testing is done according to an operational profile that has variations in frequency of application functions and when early fault corrections have a greater effect on the failure rate than later ones. Thus the failure rate has a decreasing slope. The assumptions for this model are as follows;

- Failures are independent of each other
- The failure rate decreases exponentially with execution time

Representative models in this class include Duane’s model, Brooks and Mutely Binomial and Poisson models, Yamada’s S-Shaped model and logarithmic Poisson model.

iii. Bayesian Models

A Bayesian model takes a subjective viewpoint in that if no failures occur while the software is observed then the reliability should increase, reflecting the growing confidence in the software by the user. The reliability is therefore a reflection of both the number of faults that have been detected and the amount of failure free operation. This reflection is expressed in terms of a prior distribution representing the view from past data and a posterior distribution that incorporates past and current data. NHPP models assume that the hazard function is directly proportional to the number of faults in the program and hence the reliability is a function of this fault count. The Bayesian approach argues that a program can have many faults in unused sections of the code and exhibit reliability than software with only one fault in a frequently exercised section of code. The Bayesian model also reflects the belief that different faults have different impacts on the reliability of the program. The number of faults is not as important as their impacts. If we have a program that has a numbers of faults in code, then the program is less reliable than the one
that has only one fault but which is used often. The Bayesian modeler says that it is more important to look at the behavior of the software than to estimate the number of fault in it. The mean time to failure would therefore be a very important statistic in this framework (Michael Lyu 1996). The assumption for this model is:

- Successive execution times between failures $X_i$’s are assumed to be independent exponential random variables with parameter.

- The parameters form a sequence of independent random variables, each with a gamma distribution of parameters and functions. The function is taken to be an increasing function of ‘I’ that describes the quality of the programmer and the difficulty of the task.

- The software is operated in a manner similar to the anticipated operational usage.

Representative models of this class are those developed by Bev Littlewood.

Software undergoes several stages of testing before it is put into operation. In every stage of testing, modification and corrections are made with the hope of increasing reliability (Bo Yang et al 2008). All existing software reliability models are developed for the software products that are statically constructed normally by a company or institution that has the full control of the development process. The evolutorial shift from the product-oriented software architecture to the service oriented architecture (SOA) and Web Services (WS) invalids many techniques developed for traditional software (Tsai et al 2004). Hence web based application with alternative approach for parameter estimation in Gompertz software reliability model is considered in this work.
Gompertz model is practiced for software reliability prediction with an alternative approach for parameter estimation. 50 real time projects from various IT industries are considered. It is observed that the modified approach for parameter estimation in Gompertz model would accurately predict the reliability of the software.

1.5 ORGANIZATION OF THE THESIS

The intention has been to make this report self-contained. The structure of this report is described as follows:

**Chapter 1** presents the Objective of the work, general introduction of Software Engineering, Software Reliability Engineering and Software Reliability and Models.

**Chapter 2** consists of a detailed review of literature of Software Reliability Models, Parameter estimation methods. The previous literatures about the work related are reviewed. Literature of application of various Software Reliability models is explained.

**Chapters 3 and 4** illustrate the Gompertz Software Reliability model, model descriptions and derivations and various methods of parameter estimation.

In **Chapter 5**, the results obtained from both the methods of parameter estimations and fitness analysis have been discussed. **Chapter 6** concludes the thesis with future research directions.

1.6 SUMMARY
A brief introduction is given about the software reliability which is a very active research field in software Engineering. Various software reliability models are analysed. The next chapter presents the review of the research work carried out in the area of Software reliability models.