CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

The complexity of computer systems has grown significantly during the past decades. Researchers and engineers have worked to increase the chance that the software systems will perform satisfactorily during operation. This process required the removal of faults during the testing phase. Errors are bound to happen as software is written by humans. Increase in the demand of software has led to the study of the high quality reliable software development. Reliability is the most important aspect. It measures software failures during the process of software development. It deals with probabilistic methods applied to the analysis of random occurrences of failures in a software system. A common approach for measuring software reliability is by using an analytical model whose parameters are generally estimated from available data on software failures (Lyu, 1996; Musa et al.1987). A software reliability growth model (SRGM) is a mathematical expression of the software error occurrence and the removal process. In early 1970’s, many software reliability growth models have been proposed. A Non Homogeneous Poisson process (NHPP) as the stochastic process has been widely used in SRGM.

In this study, the statistical description is concerned with the time-intervals between failures. Thus, software reliability can be defined as the probability that no failure occurs up to time t. A failure is the departure of software behavior from the user requirements. This phenomenon must be distinguished from the fault in the software code which causes the occurrence of failure as soon as it is activated during program execution. If each time after a failure has been experienced, the underlying fault is detected and fixed correctly and then the reliability of software will improve with time. Many software reliability models (SRM) are based on the inter-failure times. These successive software failure times are assumed to be statistically independent. In order to manage the projects effectively, it is hoped software is made more reliable, can be modeled through the use of Software Reliability Growth Models (SRGMs). It is now well-recognized that reliability growth models are better described by a Non-Homogeneous Poisson Process
In the past decades, various statistical tools have been applied to solve the problems of software reliability. By proposing statistical models, i.e. software reliability models, reliability can be measured.

Software reliability models are applied to help estimate the total number of faults latent in a software system and evaluate whether the software meets the reliability requirements, and provide a guiding measurement to determine the optimal release time of a software system before the software is released to customers. By estimating, predicting and managing the reliability, cost and performance of software-based systems, the quality of software products is aided to get improved and the satisfaction of customers are gained.

Software reliability models set up an effective, quantitative bridge to understand and measure software quality. Of the existing stochastic software reliability models, a large subgroup is the Non-Homogeneous Poisson Process (NHPP) software reliability growth model. It has been theoretically justified the appropriateness of using NHPP to characterize the failure process (Miller, 1984). The Goel-Okumoto model is one of the first such models (Goel and Okumoto, 1979). More on software reliability can be found in Musa et al. (1987), Xie (1991), Lyu (1996), and Pham (2000).

In this research, the NHPP software reliability models are studied. Various NHPP models have been developed based on some specific assumptions. Many models such as the G-O model assume perfect debugging, i.e. each time a failure occurs, the fault that has caused the failure is immediately and completely removed and no new faults are introduced. Another assumption of NHPP reliability models is that each failure occurs independently and randomly from the same distribution during the fault detection process (Musa et al., 1987). The failure distribution can be affected by many factors, such as the running environment, testing strategy and resource allocation. Once these factors are changed during the software-testing phase, the failure process may have changed from one distribution to another. This is the so-called change-point problem (Zhao, 1993).

The desirable characteristic of any product is the quality. The quality of software can be judged by its correctness, adaptability and testability. A software
product has a typical development process. Some activities of the process are related to particular phases of development. The other activities of the software process called umbrella activities are applied throughout the software development. One of those activities is assessing quality of the software. As in the case of any other product to obtain quality software the only way is to identify the opportunities of improvement at every stage of software development. The tests and reviews conducted throughout the development come under the quality control. A perfect program is said to be reliable while an imperfect one is said to be unreliable.

Numerous SRGMs have been proposed, and some appear to be better overall than others. Unfortunately, models that are good overall are not always the best choice for a particular data set, and it is not possible to know which model to use a priori. Even when an appropriate model is used, the predictions made by a model may still be less accurate than desired. For this reason, a great deal of research has gone into trying to make more effective use of existing models.

The goal of this thesis is to discover methods which will allow us to calculate the reliability and stability of the software using software reliability growth models. To do this we need quantitative ways of measuring the model parameters. It is to measure how well a model fits the observed data; we are concerned with a model’s ability to examine the stability of software. How the parameters should be estimated and what other techniques should be employed to ensure that traditional software reliability growth models give the best results possible. In the process of doing this we will examine exactly what the parameters of these models mean.

Quantitative approaches, particularly statistical techniques, have successfully been used to support project control and process improvement in manufacturing environments. Such approaches have also been applied within software engineering to enable a better understanding of software development and to improve software product quality. This thesis demonstrates how a sound adoption of Statistical Process Control (SPC) based control charts; order statistics and Sequential probability Ratio Test (SPRT) can form the basis for identifying the process variation using software defect data, thereby, improving software quality and reliability.
1.2 SOFTWARE RELIABILITY

The ability of a system or a component to perform its required functions under stated conditions for a specified period of time is referred to as software reliability. It is a probabilistic measure and can be defined as the probability that software faults do not cause a failure during a specified exposure period in a specified use environment. The probabilistic nature of this measure is due to the uncertainty in the usage of the various software functions and the specified exposure period, that may mean a single run, a number of runs, or time expressed in calendar or execution time units. Software reliability is a useful measure in planning and controlling resources during the development process so that high quality software can be developed.

Usually logic errors in the software are not hard to fix but diagnosing logic bugs is the most challenging. For many reasons, the fault is usually subtle. A computer system consists of two major components: hardware and software. Although extensive research has been carried out on hardware reliability, the growing importance of recent software in complex applications dictates that the major focus has shifted to software reliability. Failures of the software may result in an unintended system state, may cause property damage or destruction, people are injured or killed, and / or monetary costs are incurred. As software projects become large, the rate of software defects increases geometrically, and locating software faults is extremely difficult and costly.

Software faults are more insidious and much more difficult to handle than physical defects. In theory software can be error free, and unlike hardware, does not degrade or wear out but it does deteriorate. The deterioration here, however, is not a function of time. Rather, it is a function of the results of changes made in the software during maintenance, through correcting latent defects, modifying the code to changing requirements and specifications, environments and applications, or improving software performance. All design faults are present from the time the software is installed in the computer. In principle, these faults could be removed completely, but in reality the goal of perfect software remains elusive (Friedman and Voas, 1995). Computer programs, which vary for fairly critical
applications between hundreds and millions of lines of code, can make the wrong
decision because the particular inputs that triggered the problem were not tested
and corrected during the testing phase. Such inputs may even have been
misunderstood or unanticipated by the designer who either correctly programmed
the wrong interpretation or failed to identify the problem. These situations and
other such events have made it apparent that we must determine the reliability of
the software systems before putting them into operation.

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

Research on software reliability engineering has been conducted during
the past few decades and numerous statistical models have been proposed for
estimating software reliability (Pham 1999a, 2000). Most of the estimating models
for predicting software reliability are based purely on the observation of software
product failure, where they required a considerable amount of failure data to
obtain an accurate software reliability prediction based on NHPP Models.

1.2.1. Software Reliability: An Attribute of Software Quality

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

An important quality attribute of a computer system is the degree to which it can be relied upon to perform its intended functions. Evaluation, prediction, and improvement of this attribute have been of concern to designers and users of computers from the early days of their evolutions. Software is essentially an instrument for transforming a discrete set of inputs into a discrete set of outputs. It comprises a set of coded statements whose function may be to evaluate an expression and store the result in a temporary or permanent location, decide which statement to execute next, or to perform input/output operations. Since, to a large extent, software is produced by humans, the finished product is often imperfect. It is imperfect in the sense that a discrepancy exists between what the software can do versus what the user or the computing environment wants it to do. These discrepancies are what we call software faults. Basically, software faults can be attributed to an ignorance of the user requirements, ignorance of the rules of the computing environment, and to poor communication of software requirements between the user and the programmer or poor documentation of the software by the programmer. Even if we know that software contains faults, we generally do not know their exact identity. There are two approaches available for indicating the existence of software faults, namely program proving, and program testing.

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

1.2.2. Software Process and Software quality

Process is the transformation of a set of inputs, which can include materials, actions, methods and operations into desired outputs in the form of products, information, services or results. In each area of an organization there will be many processes taking place. Each process may be analyzed by an examination of the inputs and outputs. This will determine the action necessary to improve quality. Clearly, to produce an output which meets the requirements of the customer, it is necessary to define, monitor and control the inputs to the process (Oakland, 2007).

A software process, viewed as a system, has a development life cycle consisting of activities that include requirements definition, design, implementation and test. The requirements for a software process are specified in terms of quality characteristics specified by the customer. Process design is established by partitioning the process into meaningful activities that can be defined in terms of inputs, outputs and constraints. Implementation is accomplished by specifying the procedure for each activity and assigning responsibility for each procedure. In the manufacturing arena, it is not difficult to figure out the relationship between product quality and corresponding production process. Therefore, we can measure process attributes, work on them, improve according to the results and produce high quality products. On the other hand, software product is difficult to characterize, as it is not concrete, it is difficult to recognize the correlation between a single software process and the quality of the related software product. The software process must be assessed based on its capability to develop software that is consistent with user’s requirement. Actually,
there is no specific software measure showing the extent to which customer requirements are met. However, there are processes and products that influence production life cycle by measuring specific characteristics of these processes and products. We can have an idea about the quality of the final product. One difficulty in software production is that, there is not a repetitive production. Each product is distinct and possesses different characteristics. For this reason, it is usually not possible to form a sample of n measurements. In this case, we shall take care of each single measurement, and perform the analysis accordingly.

   Beside these general software process characteristics, software measurement is also a very complicated process. Each metric requires different measurement techniques and the reliability of the metric depends on how well the metric is defined, how properly the data collection procedures are performed and how robust the measures are with respect to varying environmental conditions. Excessive variability of target processes cause defects. Defects introduced into a product have multiple effects. They require the effort of skilled personnel to detect, remove, repair and retest to improve reliability. Defects also increase process cost, time and complexity. In addition, defects that escape detection and repair before the product is released to a customer reduce reliability of the product. Activities that decrease the introduction of defects or increase the early detection of defects are prime targets for measuring the effectiveness of the process.

   The software process has been identified as a process that is dominated by design risk or cognition because each instance of a software process produces a unique product for which quality is determined by conformance to customer requirements measured in terms of software quality characteristics. Software quality characteristics are those attributes of software that include correctness, reliability, understandability, portability, maintainability, testability, robustness, usability, cost-to-develop, and time-to develop. Adherence to customer requirements refers to the elimination of variance in quality characteristics, in other words, eliminating differences between customer expectations and the delivered software product. As these requirements vary for each customer and often conflict about a given instance of the software process. Despite the difficulties mentioned, it may be still possible to apply SPC to software process.
1.3 NHPP SRGM

The NHPP group of models provides an analytical framework for describing the software failure phenomenon during testing. They are proved to be quite successful in practical software reliability engineering (Musa et al., 1987). They have been built upon various assumptions. The main issue in the NHPP model is to determine an appropriate mean value function to denote the expected number of failures experienced up to a certain time point. Model parameters can be estimated by using Maximum Likelihood Estimate (MLE). Various NHPP SRGMs have been proposed upon various assumptions. Many of the SRGMs assume that each time a failure occurs, the fault that caused it can be immediately removed and no new faults are introduced, which is usually called perfect debugging. Imperfect debugging models have proposed a relaxation of the above assumption (Ohba, 1984; Pham, 1993).

If ‘t’ is a continuous random variable with probability density function: 
\[ f(t; \theta_1, \theta_2, \ldots, \theta_k) \]
where \( \theta_1, \theta_2, \ldots, \theta_k \) are k unknown constant parameters which need to be estimated, and cumulative distribution function: 
\[ F(t) \]
Let ‘a’ denote the expected number of faults that would be detected given infinite testing time in case of finite failure NHPP models and ‘b’ represents the fault detection rate. In software reliability, the initial number of faults and the fault detection rate are always unknown. The NHPP models are further classified into Finite and Infinite failure models. Then, the mean value function of the finite failure NHPP models can be written as:
\[ m(t) = aF(t) \]
representing the expected number of software failures by time ‘t’. The failure intensity function \( \lambda(t) \) in case of the finite failure NHPP models is given by:
\[ \lambda(t) = aF'(t) \]
which is proportional to the residual fault content (Pham, 2006).

Let \( N(t) \) be the cumulative number of software failures by time ‘t’. A non-negative integer-valued stochastic process \( N(t) \) is called a counting process, if \( N(t) \) represents the total number of occurrences of an event in the time interval \([0,t]\) and satisfies these two properties:
1. If \( t_1 < t_2 \), then \( N(t_1) \leq N(t_2) \)

2. If \( t_1 < t_2 \), then \( N(t_2) - N(t_1) \) is the number of occurrences of the event in the interval \([t_1, t_2]\).

One of the most important counting processes is the Poisson process. A counting process, \( N(t) \), is said to be a Poisson process with intensity \( \lambda \) if

1. The initial condition is \( N(0) = 0 \)
2. The failure process, \( N(t) \), has independent increments
3. The number of failures in any time interval of length \( s \) has a Poisson distribution with mean \( \lambda s \), that is,

\[
P\{N(t+s) - N(t) = n\} = \frac{e^{-\lambda s} (\lambda s)^n}{n!}
\]

Describing uncertainty about an infinite collection of random variables one for each value of ‘t’ is called a stochastic counting process denoted by \([N(t), t \geq 0]\). The process \( \{N(t), t \geq 0\} \) is assumed to follow a Poisson distribution with characteristic MVF (Mean Value Function) \( m(t) \) representing the expected number of software failures by time ‘t’. Different models can be obtained by using different non decreasing \( m(t) \). The derivative of \( m(t) \) is called the failure intensity function \( \lambda(t) \).

A Poisson process model for describing about the number of software failures in a given time \((0, t)\) is given by the probability equation.

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

Where, \( m(t) \) is a finite valued non negative and non decreasing function of ‘t’ called the mean value function. Such a probability model for \( N(t) \) is said to be an NHPP model.
1.4 ORDER STATISTICS

To improve and understand the logic behind process control methods, it is necessary to give some thought to the behavior of sampling. If the length of a single failure interval is measured, it is clear that occasionally a length will be found which is towards one end of the tails of the process’s normal distribution. This occurrence may lead to the wrong conclusion that the process requires adjustment. If a sample of four or five is taken, it is extremely unlikely that all four or five failure interval lengths will lie towards one extreme end of the distribution. If we take the average or length of four or five failure intervals, we shall have a much more reliable indicator of the state of the process. Any change in the process mean, unless it is extremely large, will be difficult to detect from individual results alone. A large number of individual readings are necessary before such a change was confirmed.

The distribution of sample means reveals the change much quicker than individuals. Therefore, on a chart for sample means, plotted against time, the change in level would be revealed almost immediately. For this reason sample means rather than individual values are used to control the centering of processes. This provides a sound basis for the Mean Value Control Chart.

A subgroup or a sample is a small set of observations on a process parameter or its output, taken together in time. The size and the frequency of sampling are the two major problems in choosing a subgroup. The smaller the subgroup, there is less opportunity for variation within it. The larger the sample size the narrower the distribution of the means and they become more sensitive to detect change (Oakland, 2007). It is understood that, in any type of process control charting system, careful selection of subgroups is very important. The software failure data is in the form of <failure number, failure time>. By grouping a fixed number of data into one, the noise values may compensate each other for that period and thus the noise inherent in the failure data is reduced to great extent (Malaiya et al., 1990).
1.5 STATISTICAL PROCESS CONTROL

It was not until the 1920s that statistical theory began to be applied effectively to quality control. A good summary of historical background of Statistical Quality Control (SQC) can be found in Duncan (1986). It is a branch of industrial statistics which includes, primarily, the areas of acceptance sampling, statistical process control (SPC), design of experiment (DOE), and capability analysis. Briefly speaking, acceptance sampling methods are used in industry to make decisions regarding the disposition of “lots”; SPC techniques are employed to monitor production processes over time to detect changes in the process performance; DOE is applied to identify specific levels of important factors that lead to optimum (or near optimum) performance; capability analysis is to assess whether or not a process is capable of meeting specification limits on key quality characteristics (Woodall and Montgomery, 1999).

As an important branch of SQC, Statistical Process Control (SPC) is a methodology that aims to provide process control in statistical terms. Since the great industrial revolution in Japan, SPC has been widely used in manufacturing industries in order to control variability and improve processes (Wu and Tian, 2005; Wu and Wang, 2007), health care (Woodall, 2006; Coory et al., 2008) and service management (Herbert et al., 2003). It is a powerful collection of problem-solving tools useful in achieving process stability and improving capability through the reduction of variability (Montgomery, 2005). Its seven major tools, often called “the magnificent seven” are:

- Histogram or stem-and-leaf plot: a graphical display of tabulated frequencies.
- Control chart: a tool for detecting the occurrence of assignable causes of process shift.
- Pareto chart: a special type of bar chart used to separate the significant aspects of a problem from the trivial ones.
- Check sheet: a simple document used to collect operating data.
- Cause-and-effect diagram: a diagram shows the causes of a certain event and identifies desirable factors leading to an overall effect.
- Scatter diagram: a plot for identifying a potential relationship between two variables.

- Defect concentration diagram: a plot to show the location of errors or defects.

Of these seven tools, control chart is probably the most technically sophisticated.

In 1920s, W. A. Shewhart was working on the idea of quality control and he brought the idea that each process is driven by forces of variation. However, variation results in loss of quality by causing inefficiency and waste. If we can understand the sources of variation, we can take necessary actions to remove inefficiency, and increase quality. If we think of variation in software industry, lines of code produced a day can be considered as a variable parameter. If the same person produces Lines of Code for the same component, the amount of time he spends will be different from one day to the other. This can be explained as the variation in a process attribute.

According to Shewhart, variation in a process has two types of causes: assignable causes and chance causes. Assignable causes appear in unexpected periods and can be fixed by immediate actions. For instance, if a new tool is being used for coding, the productivity of the coder may be lower during adaptation period. When this is realized, a training program can be implemented to improve productivity. On the other hand, chance causes are the results of the system itself (Deming, 2000) refers them as Common Causes. They are naturally existent within the defined processes and can only be avoided by performing improvement programs. If we think of a software engineering firm that has no reusable code library, we witness that similar code pieces are written separately in each new application and this causes delays due to rework. Such a chance cause can be prevented by creating a reusable code library.

If the variation in the behavior of a process is predictable in statistical terms, that process is said to be in control. This means that, we can expect (within certain limits) what the outcome will be the next time we perform the same process. In this way, we can prepare more accurate project plans; do better cost estimations and schedule activities in more reasonable basis. In order to calculate
the variance in process behavior, several attributes or variables representing the outcomes of the process shall be defined.

The number of defects found during unit testing, the number of requirements that are changed after requirements analysis phase, amount of CPU utilized to perform a specific application, may all be used to understand the behavior of the processes they represent. The variability in process behavior, then, can be tracked through these measures. The aim of Statistical Process Control is: firstly, to detect assignable causes of variation in the processes and provide process control; secondly, to enable monitoring of the improvement in processes (that are already under statistical control) by demonstrating the chance causes; and Shewhart Control Charts are a good means to achieve Statistical Process Control.

During his studies at Bell Labs in 1920s, Shewhart proposed that it is possible to define limits within which the results of routine efforts must lie to be economical. Deviations in the process outcomes resulting in values out of these limits indicate that the process is not performed economically. In order to detect assignable causes, Shewhart utilized statistical quality control charts (Shewhart, 1939). Specifying control limits is an important decision for designing a control chart. Before explaining how to specify control limits, two types of error should be introduced first:

- Type I error: the risk of a point falling beyond control limits when the process is in control.
- Type II error: the risk of a point falling between control limits when the process is out of control.

When control limits are far from the centre line, Type I error is small but Type II error is large. When control limits are near to the centre line, Type II error is small but Type I error is large. As a result, optimal control limits are determined based on the tradeoff between Type I error and Type II error. In practice, k is set to be three so that 99.73% of the data will be within control limits when the process is in control and data is normally distributed. Analyzing patterns on control charts can help us to identify problems for the process.

The lower and upper control limits are derived and data is analyzed using the statistical evidence on hand. By analyzing the data values with respect to
upper and lower control limits together with their location in the zones, assignable causes are detected. Then necessary actions are taken and measurements are repeated. The charts are redrawn with the existing data values, and this process is repeated until no evidence remains for the existence of assignable causes.

Once the process is brought under control, further improvement activities are implemented to minimize the effect of common causes. The measurement can be performed by means of either variables or attributes. A variable as “measure of a product that can have any value between the limits of the measurement”, while an attribute as “count of things which may or may not be present in the product” (Burr and Owen, 1996). The nature of these two measurement categories necessitates different statistical analyses. A variable normally has normal probabilistic distribution, where as it is likely to be binomial for an attribute. Statistical analysis may be performed by implementing X and R charts for sample data. When variable measures are individual data points, Individuals Charts are mostly utilized.

1.6 SEQUENTIAL PROBABILITY RATIO TEST

Sequential analysis was first developed by Abraham Wald in the 1940’s and (Wald, 1945; Wald and Wolfowitz, 1948) introduced the SPRT during that time. The SPRT was initially developed by Wald (1947) for quality control problems during World War II. Neyman and Pearson (1933) result inspired Wald to reformulate it as a sequential analysis problem. The original development of the SPRT is used as a statistical device to decide which of two simple hypotheses is more correct. It has been formulated for use in the computerized testing of human examinees as a termination criterion. The properties of it have been studied intensively by many researchers since Wald (1947). The likelihood based SPRT proposed by Wald is very general in that it can be used for many different probability distributions. Some other people worked on this are Siegmund (1986) and Ghosh and Sen (1991).
1.7 MODEL DESCRIPTION: INFLECTION S-SHAPED MODEL

Software reliability growth models (SRGM’s) are useful to assess the reliability for quality management and testing-progress control of software development. They are commonly designed to be used with data collected in terms of the testing time between failures, and it is on such models that this study will focus. They have been grouped into two classes of models concave and S-shaped. The most important thing about both models is that they have the same asymptotic behavior, i.e., the defect detection rate decreases as the number of defects detected (and repaired) increases, and the total number of defects detected asymptotically approaches a finite value. These models are usually stated in terms of two equations; the mean value function and the failure intensity function. It is worth noting that the second function is the derivate of the first.

The inflection S-shaped model was proposed by Ohba in 1984. This model assumes that the fault detection rate increases throughout a test period. The model has a parameter, called the inflection rate, that indicates the ratio of detectable faults to the total number of faults in the target software. Regardless of how these models were originally formulated, the parameters of these models are given as ‘a’ and ‘b’. Standard practice is to determine the values of these parameters by fitting the model in question to the available data. Once the model has been fitted to the data, it can then be used to obtain estimates of current stability of the software and make predictions about the programs future reliability. True, sustained exponential growth cannot exist in the real world. Eventually all exponential, amplifying processes will uncover underlying stabilizing processes that act as limits to growth. The shift from exponential to asymptotic growth is known as sigmoidal, or S-shaped, growth.

The following paragraphs describe the model and will focus on the different functions used in estimating the reliability and stability of the process.

The inflection S-shaped NHPP software reliability growth model is known as one of the flexible SRGMs that can depict both exponential and S-shaped growth curves depending upon the parameter values (Ohba, 1984). The model has been shown to be useful in fitting software failure data. Ohba proposed that the fault removal rate increases with time and assumed the presence of two types of
errors in the software. He models the dependency of faults by postulating the following assumptions:

- Some of the faults are not detectable before some other faults are removed.
- The detection rate is proportional to the number of detectable faults in the program.
- Failure rate of each detectable fault is constant and identical.
- All faults can be removed.

Assuming [Ohba 1984]:

\[
b(t) = \frac{b}{1 + \beta e^{-bt}}
\]

This model is characterized by the following mean value function:

\[
m(t) = \frac{a}{1 + \beta e^{-bt}} \left(1 - e^{-bt}\right)
\]

where ‘b’ is the failure detection rate, and ‘\(\beta\)’ is the inflection factor. The failure intensity function is given as: \(\lambda(t) = \frac{abe^{-bt}(1 + \beta)}{(1 + \beta e^{-bt})^2}\). The model is identical with the Exponential SRGM, if the inflection rate equals 1.

1.8 FAILURE DATA ANALYSIS

In software reliability modeling, depending on the format in which test data are available, there are two common types of failure data: time-domain (i.e. ungrouped) data and interval-domain (i.e. grouped) data (Pham, 2006). These data are usually used by practitioners when analyzing, assessing and predicting reliability applications.

Type I Data---inter-failure time data, i.e. the time between two failures. In this type of data, failure data are collected chronologically and the occurrence times are recorded as \(0 \leq t_1 \leq t_2 \leq \cdots \leq t_n\), which are the realizations of the random variables \(0 \leq T_1 \leq T_2 \leq \cdots \leq T_n\). The inter-failure time is given by \(x_i = t_i - t_{i-1}\). In some literature, this type of data is also called as ungrouped data;

Type II Data---cumulative count of failures per interval, i.e. in this type of data, a period of failure observing time is divided into a set of sub-intervals, for instance \((0, t_1), (t_1, t_2) \cdots (t_{i-1}, t_i)\), and the total number of failures in the above each
sub-interval is recorded as $n_1, n_2, \ldots, n_k$. In some literature, this type of data is also called as grouped data.

In this research as Type I data is used, it is well known that for a NHPP the joint distribution of the failure times of $n$ failures that have occurred during the period of $(0, t_n)$ has the same distribution as the order statistics of a random sample with size $n$, therefore the log likelihood function takes on the following form (Pham, 2006):

$$L(\Theta, t_1, t_2, \ldots, t_n) = e^{-m(t_n)} \prod_{i=1}^{n} \lambda(t_i)$$

$$\ln L(\Theta, t_1, t_2, \ldots, t_n) = \sum_{i=1}^{n} \log[\lambda(t_i)] - m(t_n)$$

More discussions regarding the MLE issue can be found in Knafl (1992), Hossain and Dahiya (1993), Zhao and Xie (1996) and Knafl and Morgan (1996).

The estimation of model’s parameters is an important issue too. The maximum likelihood estimation technique is most commonly used in the estimation of parameters of the NHPP models. With respect to the model’s parameters, the partial derivatives of the parameters respectively are taken. Let the equations equal to zero, solve the equations and obtain the values of the parameters.