CHAPTER 1
SOFTWARE QUALITY - SOME PRELIMS

1.1 Introduction

Society is becoming ever more dependent upon systems that rely importantly upon the reliably correct functioning of software and software-controlled systems and there is always a need of high quality software. Unlike hardware, software does not age, wear out or rust. Unreliability of software is mainly due to bugs or design faults in the software (Roger Pressman). Software reliability is dynamic & stochastic. The exact value of product reliability is never precisely known at any point in its lifetime. The study of software reliability can be categorized into three parts: Modeling, Measurement & improvement. Many Models exist, but no single model can capture a necessary amount of software characteristics. There is no single model that is universal to all the situations. Simulations can mimic key characteristics of the processes that create, validate & review documents & code. Software reliability measurement is naive. It cannot be directly measured, so other related factors are measured to estimate software reliability. Software reliability improvement is necessary & hard to achieve (Aasia Quyoum, Mehraj-Ud-Dindar, S.M.K.Quadri, 2010).

The reliability of software can be improved by understanding the characteristics of software and its design. Complete testing of the software is not possible; however sufficient testing & proper maintenance will improve software reliability to great extent. As the functionality of computer operations becomes more complicated and increase in size and complexity, there is much necessity for looking at ways to quantify and predict the reliability of computer systems in various complex operating environments (Pham, 2005c).

Software reliability is defined as the probability of failure free software operation for a specified period of time in a specified environment (Musa, 1998). There are basically two approaches for performing statistical reliability prediction for software. The first approach is based on design parameters, estimates the number of defects in the software using code characteristics such as number of lines of code, nesting of
loops, external references, input/output calls, etc. The second approach is reliability growth analysis based on statistical correlations of actual defect detection data obtained during testing. Many models are used to describe reliability growth in software such as Crow-AMSAA, standard Gompertz, modified Gompertz and Lloyd-Lipow (Reliability Hotwire, 2008).

In our modern society, the usage of computers is getting increased day by day and is used in diverse areas. As computer applications became more diverse and spread through almost every area of everyday life, reliability became a very important characteristic for software, since it is a matter of economy. To produce software with high reliability, it is necessary to measure and control its functionality. To make better predictions number of models can be developed.

The demand for complex hardware/software systems has increased more rapidly than the ability to design, implement, test, and maintain them. When the requirements for and dependencies on computers increase, the possibility of crisis from computer failures also increases. The impact of these failures ranges inconveniences from economic damage (e.g., interruptions of banking systems), to loss of life (e.g., failures of flight systems or medical software). Needless to say, the reliability of computer systems has become a major concern for our society. When we entered the last decade of the 20th century, computer software had already become the major source of reported outages in many systems (Gray, 1985). Consequently, recent literature is replete with horror stories of projects gone away, generally as a result of problems traced to software.

For a large scale or international company, successful development of a software system depends on its software components. Therefore, the reliability of a large software system needs to be modelled analyzed during the software development process. The future failure behavior of a software system is predicted by studying and modelling its past failure behavior (Yamada, 1993 and 1985, Musa, 1998). It is very important ensure the quality of the underlying software systems in the sense that they perform their functions correctly. As a result software companies see a major share of project development costs identified with the design, implementation, and assurance of reliable software, and they recognize a tremendous need for systematic approaches using software reliability engineering techniques. Clearly,
developing the required techniques for software reliability engineering is a major challenge to computer engineers, software engineers, and engineers of various disciplines for now and the decades to come. This foundational role means that the reliability of systems software is of primary importance which made me to do research in assessing the reliability of software with a software reliability growth model.

Software reliability is the application of statistical techniques to data collected during system development and operation to state, forecast, estimate, and assess the reliability of software-based systems (DAC, 2006). There are several reasons why software reliability is required and is important. In general, the unreliable software is likely to be discarded by the users and any company producing unreliable software will definitely loses its reputation. Also the failure cost of the software is really high and it is quite difficult to improve. The systems which are not reliable may lead to information loss which is sometimes very important. The Reliability of a software system is a measure of how well users think it provides the services that they require. Software reliability is a function of the number of failures experienced by a particular user of that software. So it is claimed that software installed on an aircraft will be 99.99% reliable during an average flight of five hours. This means that a software failure of some kind will probably occur in one flight out of 10,000.

Software reliability modelling and prediction during product development is an area of reliability that is getting more focus from software developers. Many models have been developed with different mathematical techniques to adapt to different testing environments (Lyu, 1996). The reliability measurements are crucial for the management to make decisions, such as testing – resource allocation (Yamada et al., 1995), test stopping decision (Littlewood and Wright, 1997; Xie and Hong, 1999) and cost analysis (Xie and Yang, 2003). Generally these models can be categorized into two groups: data-driven software reliability models (Xie, 1991; Pham, 2000) and analytical software reliability models (Gokhale and Trivedi, 1999). The use of software reliability growth models plays an important role in measuring improvements, achieving effective and efficient test scheduling during the course of a software development project, determining when to release a product or estimating
the number of service releases required to reach a reliability goal. Reliability is a real-world phenomenon with many associated real-time problems.

One of the common practices in manufacturing industry is Statistical Process Control (SPC). The investigation on quantitative mechanisms as an aid to control process variation led to the application of the SPC since 1930’s. The idea of applying SPC to software development however, is exemplified mainly by Capability Maturity Model (CMM) in mid 90’s. Although its benefits are accepted for manufacturing companies, there have been many debates about its application in software development (Komil Umit Sargut, 2003).

Data-driven models focus on the failure data generated through the software testing process. They consider software reliability prediction as a time-series analysis problem. These models are developed from past software failure data and have less restrictive assumptions. Analytical software reliability models describe the software failure behaviour during the software testing process and take this process as a stochastic process. For these models, some restrictive assumptions are made such as perfect and immediate fault correction. According to different modelling techniques, these models can be grouped into NHPP models, Markov models and Bayesian models (Hu, 2006). Among these three models, due to flexibility and simplicity NHPP models are applied extensively, which are the main concern for the present thesis.

The cost to remove a defect will increase radically with the development process. Therefore, it is essential for the management to make timely and cost effective decisions to keep the process under control. From the perspective of the software lifecycle, the trend is to remove the software defects and evaluate the software reliability. In the analysis phase, testing is conducted at the architecture level, with either state-based or path-based models (Goseva-Popstojanova and Trivedi, 2001; Cukic, 2005; Wang et al., 2006). In the testing phase, the reliability of the software improves through debugging. A reliability growth model is needed to estimate the current reliability level, the time and resources required to achieve the objective reliability level. During this phase, reliability estimation is based on the analysis of failure data. The number of failures experienced can be denoted as a stochastic
counting process characterized by its mean value function. This process can be represented by a Poisson model.

Research activities in software reliability engineering have been conducted and a number of NHPP software reliability growth models have been proposed to assess the reliability of software. Software reliability can be estimated once the mean value function is determined. The technique of control chart has been used in the software engineering so as to improve the quality of software products.

A work applying statistical quality control in measuring software reliability is carried by Stieber (1997). The Sequential Probability Ratio Test developed by Wald (1947) assumes that software failure follows a Homogenous Poisson Process (HPP). But, most cases faced in reality are non-homogenous. Under this consideration, the failure process is monitored both on Time domain data and Interval domain data.

The following sections of this chapter discuss concepts related to the areas of Non-Homogeneous Poisson Process (NHPP), Software Reliability Growth models (SRGMs), Statistical Process Control (SPC), Maximum Likelihood Estimation (MLE) and Sequential Probability Ratio Test (SPRT) which are extensively used in carrying out this thesis.

**Non-Homogenous Poisson Process (NHPP)**

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. This section discusses stochastic reliability models for the software failure phenomenon based on NHPP. An NHPP is a realistic model for assessing software reliability and has a very interesting and useful interpretation in debugging and testing the software. There are two main types of software reliability models: the deterministic and the probabilistic. Performance measures of the deterministic type are obtained by analyzing the program texture and do not involve any random event. Two well-known models are: McCabe’s Cyclomatic complexity metric (McCabe, 1976) and Halstead’s software metric (Halstead, 1977). The probabilistic model represents the failure occurrences and the fault removals as probabilistic events. The probabilistic software reliability models can be classified into different groups
(Pham, 2000) such as, Error seeding, Curve fitting, Failure rate, Reliability growth, Markov structure, Time-series and NHPP. In this thesis, NHPP type of software reliability models and methods are used for estimating software reliability.

A process which develops in time in accordance with some probabilistic laws is called a stochastic process. They are used for the description of a systems operation over time. The two main types of stochastic processes are continuous and discrete. Counting processes in reliability engineering are widely used to describe the appearance of events in time, e.g., failures, number of perfect repairs, etc. The simplest counting process is a Poisson process. Poisson-type models assume that the number of failures detected within distinct time intervals is independent.

1. Homogeneous Poisson Process (HPP): with the same rate of failure.

**NHPP Models**

The NHPP group of models provides an analytical framework for describing the software failure phenomenon during testing. These models are usually based upon various debugging scenarios, and can catch quantitatively typical reliability growth observed in the testing phase of software products. The NHPP based SRGMs are proved to be quite successful in practical software reliability engineering (Musa et al., 1987). Many of the SRGMs assume that each time a failure occurs, the fault that caused it can be immediately removed and new faults are not introduced. It is usually called perfect debugging. Imperfect debugging models have proposed a relaxation of the above assumption (Pham, 1993).

If ‘t’ is a continuous random variable with probability density function (pdf):
\[ f(t, \theta_1, \theta_2, ..., \theta_k) \]
and cumulative distribution function (cdf): \[ F(t) \]. Where, \( \theta_1, \theta_2, ..., \theta_k \) are k unknown constant parameters. The mathematical relationship between the pdf and cdf is given as: \[ f(t) = F'(t) \].

Let \( N(t) \) be the cumulative number of software failures by time ‘t’. A nonnegative 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 \( t_2 < t_1 \), 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 ‘\( \lambda s \)’.

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

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 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 \( m(t) \).

If future changes in \( N(t) \) are independent of the past changes in it, the probability distribution of \( N(t) \) in a time interval say \([t_1, t_2]\) depends only on the length \( t_2 - t_1 \) but not on the extremities \( t_1, t_2 \), as a constant multiplier of ‘t’ i.e., \((t_2 - t_1) \) say \( mt \), then the probability distribution of \( N(t) \) for any non-negative ‘t’ can be derived as

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

This is called a Homogeneous Poisson Process (HPP).

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

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

The mean value function \( m(t) \) is the characteristic of the NHPP model. There are two major classes of \( m(t) \) used to describe different processes: increasing concave and S-shaped models (Ohba, 1984b; Yamada et al., 1984). A concave \( m(t) \) describes the fault detection process with exponential decreasing intensity. S-shaped \( m(t) \) describes fault detection process with increasing-then-decreasing intensity. The derivative of \( m(t) \) is called the failure intensity function \( \lambda(t) \) which is proportional to the residual fault content. In NHPP SRGM, the failure intensity \( \lambda(t) \) is proportional to the residual fault content \( [a(t) - m(t)] \) and can be expressed in general as \( \lambda(t) = b(t)[a(t) - m(t)] \). Where \( a(t) \) is the time-dependent fault content function which includes the initial and introduced faults in the program, \( b(t) \) is the time-dependent fault detection rate. A constant \( a(t) \) implies the perfect debugging assumption. A constant \( b(t) \) implies the imperfect debugging assumption (Gokhale and Trivedi, 1999; Pham, 2007).

The NHPP models are further classified into Finite and Infinite failure models. Let ‘a’ denote the expected number of faults that would be detected if infinite testing time is given in case of finite failure NHPP models. Then, the mean value function of the finite failure NHPP models can be written as: \( m(t) = aF(t) \). The failure intensity function \( \lambda(t) \) is given by: \( \lambda(t) = aF'(t) \) (Pham, 2006).

One simple class of finite failure NHPP model is the Goel and Okumoto model (Goel and Okumoto, 1979), which has an exponential growth of the cumulative number of failures experienced. Its generalization, the model under consideration in the thesis is the Burr Type XII also belongs to this class.

1.2 Software Reliability

Software Reliability is one of the most important aspects of software quality. Software Reliability Engineering (SRE) is the quantitative study of the operational behaviour of software-based systems with respect to user requirements concerning reliability (IEEE, 1995). According to ANSI, Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified
environment (ANSI/IEEE, 1991; Lyu, 1996). Software Reliability is defined as a probabilistic function, and comes with the notion of time. A certain user profile may function flawlessly while a different one may fail and we cannot conclude that software has zero reliability (Ansi 1993) (Lyu 1995). Mathematically, reliability \( R(t) \) is the probability that a system will be successful in the interval from time ‘0’ to time ‘\( t \)’:

\[
R(t) = P(T > t) \quad t \geq 0
\]

Where, ‘\( T \)’ is a random variable which denotes the time-to-failure.

Software reliability models serve to aid the software engineer by indicating the likelihood of system operation over a given time interval according to the stated specifications. Software reliability predictions can be used to judge the quality of a program. It is important to define the key terms in developing and describing software reliability models. The following definitions are used to distinguish between failures and faults throughout the thesis (Musa et al., 1987).

A failure is a departure from how software should behave during operation according to the requirements.

A fault is a defect in a program, that when executed causes failure(s).

The study of software reliability can be categorized into three parts: modelling, measurement and improvement. There are two general schemes to demonstrate whether the required software reliability has been achieved or not. They are formal verification and statistical testing. Various approaches to estimate software reliability by statistical testing is reviewed in this section.

Both static and dynamic software reliability models exist to assess the quality aspect of software. A static model uses software metrics, like complexity metrics, results of inspections, etc., to estimate the number of defects or faults in the software. Dynamic models use the past failure discovery rate during software execution or cumulative failure profile to estimate the number of failures. Because of this, they include a time component, which is typically based on recording times ‘\( t_i \)’ of successive failure ‘\( i \)’ (\( i > 0 \)). Time may be recorded as execution time or calendar
Software modelling techniques can be divided into two subcategories. They are prediction modelling and estimation modelling (RAC, 1996). Both kinds of modelling techniques are based on observing and accumulating failure data and analyzing with statistical inference.

There are different approaches to modelling software reliability such as, Reliability growth models, Coverage-based models and Component-based models. In this thesis reliability growth models are focused. There is considerable statistical literature on modelling the reliability growth process of finding and fixing defects in a software product.

Most software models contain assumptions, factors, and a mathematical function that relate the reliability with the factors. The mathematical function is usually exponential or logarithmic. Models differ based on their assumptions of the software and its execution environment. Some models use an NHPP model to the failure process.

SRGMs are mathematical models that represent software failures as a random process and can be used to evaluate development status during testing. Many SRGMs have been proposed in the past four decades (Xie, 1991) for the estimation of reliability growth of products during software development processes. In general, SRGMs are applicable to the late stages of testing in software development.

A number of analytical models proposed to address the problem of software reliability measurement can be classified into four main groups, according to the nature of the failure process studied. They are Input domain based models, Fault seeding models, Time between failures models and Failure count models. Among these four categories of models the focus is on those which deal with the inter failure time (i.e., Time between failure models) and the random number of software failures in a given period of time (i.e., failure count models) of a developed software. The focal theme of the thesis is to study the quality and reliability of a software product using SPC and SPRT.
Time Domain / Times Between Failure (TBF) models:

The preferred data in TBF models is the observed running times between successive failures. It is assumed that the time between two consecutive failures follows a distribution whose parameters are dependent upon the number of defects remaining in the software during that time interval. These distribution parameters are to be estimated from the observed TBFs data.

In this class of models the time between failures is a random variable following a certain probability distribution. Specifically if \( T_i \) denotes the time between \((i - 1)^{th}\) and \(i^{th}\) failure, modelling will be done on the random variable \( T_i \). The distribution of \( T_i \) reflects the improvement in the software quality as faults are detected and removed from the software. The basic model of this category is due to Jelinski and Moranda (1972). Some modifications of this model are subsequently given by Schick and Wolverton (1973) and Goel and Okumoto (1978). The model that falls in this category but suggested through Bayesian approach is that of Littlewood and Verral (1973). Another model of this category through notion of geometric deeutrophication is given by Moranda (1975). The key assumptions of this type of model are given as follows (Pham, 2006).

- Time between failures is independent.
- The exposure of each fault is of equal probability.
- Independence of embedded faults.
- Faults are removed after the occurrence of each failure.
- New faults are not introduced during correction.

Interval Domain / Failure count models:

In these models, the variable of interest is the number of failures observed per specified time interval. The idea is that the number of failures observed per time interval can be modelled according to a Poisson process. The Poisson distribution is widely used to model the number of occurrences of some event in a time interval.

The failure counts are assumed to follow a known stochastic process with a time dependent discrete or continuous failure rate. As faults are removed from the system it is expected that the observed number of failures per unit time will decrease. Here,
the time metric can be calendar time or CPU time. The time intervals are fixed and observed number of failures in each interval is treated as a random variable. The pioneering models of this category are due to Goel and Okumoto (1979), Yamada et al. (1983) and Musa and Okumoto (1984). The key assumptions of this type of model are given as follows (Pham, 2006).

- Testing intervals are independent of each other.
- Testing within intervals is reasonably homogeneous.
- Number of faults detected during non overlapping intervals is independent of each other.

Because of the data requirements for both time domain and interval domain models, considerable testing must be performed to estimate the parameters.

There are several functions of fundamental importance in modern reliability engineering (Lewis, 1996). The first and fundamental function of importance is the density function. For a continuous variable, the density function is denoted by \( f(t) \), gives the relative frequency with which the t-values occur. Characteristic of these density functions is the fact that \( \int_D f(t)dt = 1 \) for the continuous case. Here, ‘D’ denotes the domain of definition or interval of integration. All other functions considered depend on the density function and its characteristics.

The second important function from the estimation and interpretation standpoint is the cumulative density function and is denoted by \( (t) \).

Which is given as \( F(t) = \int_{t_0}^t f(t)dt \) where, ‘\( t_0 \)’ is the lower limit of domain ‘D’.

1.3 Statistical Process Control (SPC)

Statistical Process Control (SPC) is an analytical decision making tool for monitoring and controlling manufacturing processes. Walter Shewhart is the founder father who created the method of SPC during the 1920s. SPC determines when a statistically significant change has taken place in the process or when a seemingly significant change is just due to chance causes. During his investigation for the production of faulty handsets, he found out that Bell Telephones Company continuously changed its production line. Each change resulted in the production of different formats of handset telephones that made difficulty in differentiating them.
Shewhart through his research framed manufacturing problems as ‘assignable cause’ and ‘chance cause’ and introduced the SPC chart to differentiate the two. We refer to these statements/problems as predictable (normal cause variation) and unpredictable (special cause variation). As a result Bell Telephones understood the importance of reducing the variation in the manufacturing process to ensure lean production. SPC tools can be better adapted to a software product also with necessary amendments (John Oakland, 2008).

First control charts were developed by the communication industries in 1920s. In 1930s many other industries adopted the principles of SPC. America ignored the principles of SPC, where as Japan began widely using them in 1950s. Later, in 1980s Ford Motor company adopted SPC principles due to pressure on the US Market from Japanese imports. Besides in 2000 SPC was adopted in the National Health Service (NHS) too. Data should always be presented in a way that preserves the evidence. Shewhart (1931) suggested that displaying data using averages and aggregates loses the richness of the individual data points. SPC displays the individual data points (in the NHS these are often individual patients), then provides analysis to interpret what the user sees.

The application of SPC for improving software quality has been accelerating for the past few years. The quest for an improved competitive position is greatly enhanced by focusing on the process rather than the product. For effective improvements in quality, however, statistical methods provide effective and efficient guidelines. Statistical control in relation to process control charts is explained in detail in this section.

A process is the transformation of a set of inputs into desired outputs in accordance with customer specifications (Lantzy, 1992). Each process may be analyzed by an examination of the outputs. This determines the action necessary to improve quality.

The software engineering process can be defined as a set of software engineering activities needed to transform user requirements into software (Humphrey, 1989). All processes can be monitored and brought ‘under control’ by gathering and analysing data.
SPC is the application of statistical methods to the monitoring and control of a process to ensure that it operates at its full potential to produce conforming product. It is a collection of problem-solving tools which help in achieving process stability through the reduction of variability (Montgomery and Woodall, 1997). The basic SPC tools are, process flowcharting, histograms, check sheets, graphs, cause and effect analysis, scatter diagrams and control charts (Mitra, 2001). The most popular technique for maintaining process control is control charting. SPC is about using control charts to manage software development efforts in order to improve software process.

**Control charts**

Control charts, also known as Shewhart charts (Nelson, 1984) or process behaviour charts. A control chart is a specific kind of run chart that allows considerable change to be differentiated from the natural variability of the process. They separate common from special variation. They are graphical tools that help people to study the type and amount of variation present in a system. They can help identify special or assignable causes for factors that impede peak performance (Walter A. Shewhart). A control chart consists of:

1) Data points are either averages of subgroup measurements or individual measurements plotted on the x/y axis and joined by a line. Time is always on the x-axis.
2) The Average or Centre Line is the average or mean of the data points and is drawn across the middle section of the graph, usually as a heavy or solid line.
3) The Upper Control Limit (UCL) is drawn above the centre line and often annotated as “UCL”. This is often called the “+ 3 sigma line”.
4) The Lower Control Limit (LCL) is drawn below the centre line and often annotated as “LCL”. This is called the “- 3 sigma line”.
Figure 1.3.1: Example of a shewhart SPC chart.

CL= Center Line, LCL= Lower Control Limit, UCL= Upper Control Limit.

The run chart becomes a control chart if decision lines are added. Control limits are mathematical limits used to interpret the pattern on a control chart. They are derived from knowledge of distribution theory and are calculated numbers in relationship to the data. They are boundaries of expected or common variation. A statistically controlled process will oscillate fairly and evenly about the mean within the limits.

A control chart illustrates the dynamic performance of the process. It assists the software development team to identify failures and actions to be taken during software failure process which assures better software reliability. Few researchers proposed SPC based software reliability monitoring techniques (Card, 1994; Florac et al., 2000; Jalote, 2002; Baldassarre et al., 2004; Caivano, 2005).

The x and y axes should be labeled and a title need to be specified for the chart. Basically there are two categories of Control Charts: Attribute and Variable (Wheeler, Donald J. and Chambers, David).
Attribute Control Charts

Attribute Control Charts are used to monitor an organization's progress at removing defects that are inherently present in a process. Attribute charts are based on data that can be grouped and counted as present or not. Attribute charts are also called count charts and attribute data is also known as discrete data. Attribute data is measured only with whole numbers. Examples include:

- Acceptable vs. non-acceptable
- Forms completed with errors vs. without errors
- Number of prescriptions with errors vs. without

Examples of Attribute Control Charts include (Quality one.com):

- p chart: Proportion of nonconforming units in a sample.
- np chart: Number of nonconforming units in a sample.
- c chart: Number of defects per unit where the sample size is 1.
- u chart: Average number of nonconformities per unit.

Defects must exist for Attribute Control Charts to be effective and are meant to be prevented, not measured. If defects are present, Attribute Control Charts can be used to track progress. Variables Control Charts should eventually replace Attribute Control Charts.

Variable Control Charts

Variable charts are based on variable data that can be measured on a continuous scale. Variables Control Charts monitor process parameters or product features. A variable's measurement can indicate a significant change in process performance without producing a non-conformance. As a result, Variables Control Charts are more sensitive to change and are more efficient than Attribute Control Charts. A change in process behavior signals a desired or undesired need for investigation and action. Two primary statistics are measured and plotted on a Variables Control Chart: central tendency and process dispersion. Central tendency is the process Mean or X Bar. Dispersion is either the Standard Deviation or Range.
Examples of Variables Control Charts include:

X Bar and R consist of a pair of charts:
- One to monitor the process Standard Deviation, as approximated by the sample moving range.
- One to monitor the process Mean.
- X Bar and R charts plot the Mean value for the quality characteristic across all units in a sample and the range of the quality characteristic across all units in the sample.

IMR (Individual Moving Range) consists of a pair of charts:
- The individuals chart, displays the individual measured values.
- The moving Range chart displays the difference from one point to the next.

Multi-Variable Chart:
- Similar to X Bar and R except multiple characteristics are plotted against each other.
- The assumption is that the two or more characteristics are related in some manner.

SPC is used to monitor the performance of a software process over time in order to verify that the process remains in the state of statistical control. It helps in finding assignable causes, long term improvements in the software process. Software quality and reliability can be achieved by eliminating the causes or improving the software process or its operating procedures (Kimura et al., 1995).

The control charts can be classified into several categories, as per several distinct criteria. Depending on the number of quality characteristics under investigation, charts can be divided into univariate control charts and multivariate control charts. Furthermore, the quality characteristic of interest may be a continuous random variable or a discrete attribute. Control charts should be capable to create an alarm when a shift in the level of one or more parameters of the underlying distribution or a non-random behaviour occurs. Normally, such a situation is reflected in the control chart by points plotted outside the control limits or by the presence of specific patterns. The most common non-random patterns are cycles, trends, mixtures and stratification (Koutras et al., 2007). For a process to be in control the control chart should not have any trend or non-random pattern (Naikan, 2008).
Data forms the basis for analysis, decision and action. Their form and presentation will obviously differ from process to process. After the data is collected, they are analyzed and useful information is extracted through the use of statistical methods. If data is not carefully and systematically recorded, especially at the point of operation, they cannot be analyzed and put to use.

The selection of proper SPC charts is essential to effective statistical process control implementation and use. The SPC chart selection is based on data, situation and need (MacGregor and Koutri, 1995). Many factors influence the process, resulting in variability. The causes of process variability can be broadly classified into two categories, viz., Assignable (special) causes and Chance (common) causes (Shewhart, 1931; Deming, 1986).

Chance causes of variation are always present and influence the performance of the process. They are innate to the production process. Variation due to only chance causes should not be reacted to, unless it is to fundamentally improve the process. Unnecessary intervention may induce extra variation and result in economic loss. Assignable causes are occasional disturbances to the process that interfere with its common cause operation. They may arise periodically in a somewhat unpredictable fashion and cause problems. They should be detected and removed to bring the process back in its most economic state.

Software process control is used to secure the quality of the final product which will conform to predefined standards. In any process, regardless of how carefully it is maintained, a certain amount of natural variability will always exist. A process is said to be statistically “in-control” when it operates with only chance causes of variation. On the other hand, when assignable causes are present, the process is said to be statistically “out-of-control.”

The control limits are utilized to monitor the failure times. After each failure, the data can be plotted on the chart. If the plotted point falls between the calculated control limits, it indicates that the process is in the state of statistical control and no action is warranted. If the point falls above the UCL, it indicates that the failure occurrence rate may have decreased which results in the increase of TBFs. This is an important indication of possible process improvement. If the plotted point falls
below the $LCL$, it indicates that the failure occurrence rate may have increased which results in the decrease of failure time. This means that process may have deteriorated.

The original Shewhart charts are mostly useful for larger changes in performance. They have been developed for all well known probability distributions. The X-chart and Xbar-charts are used respectively for individual observations and means of subgroups of normally distributed data. These charts are also suggested for data with unknown distributions (Wheeler, 1995). Additional supplementary runs rules (Wheeler, 1995; Montgomery and Woodall, 1997; Koutras et al., 2007) have been developed to detect smaller changes. Other types of SPC charts are the cumulative sum and exponentially weighted moving average control charts. They are better at detecting smaller changes in performance than the supplementary runs rules (Hawkins and Olwell, 1998).

The success of SPC charts depends on the use of rational subgroups (Wheeler, 1995). A rational subgroup is set of observations that were generated under rather identical conditions. Out-of control situations are more likely to appear between rational subgroups and therefore typically affect all or none of the observations in a rational subgroup. SPC charts should be used with observations that are closely related to the origins of variation (Montgomery and Woodall, 1997).

It is typically recommend for a Shewhart Xbar-chart to have at least 25 to 30 subgroups of 4 or 5 observations each to obtain reliable estimates of the variability and hence to set up control limits (Quesenberry, 1997; Luke et al., 1993; Jacob and Sreejith, 2008). This is considerably more for Shewhart individual charts (Quesenberry, 1993) and Cusum charts (Hawkins and Olwell, 1998).

SPC charts help to judge the state of the process correctly through the use of properly calculated control limits. They are based on the distribution of the rationally sampled observations and the cost of type I and II decision errors. The control limits for the chart are defined in such a manner that the process is considered to be out of control when the time to observe exactly one failure is less than $LCL$ or greater than $UCL$. A type I decision error (i.e. false positive) is made when the process is judged to be out-of-control when in fact it is in-control. A type II decision error (i.e., false
negative) is made when the process is judged to be in-control when in fact it is out-of-control (Vanbrackle and Williamson, 1999; Naikan, 2008). Our aim is to monitor the failure process and detect any change of the intensity parameter. When the process is normal, there is a chance for this to happen and it is commonly known as false alarm. The traditional false alarm probability is to set to be 0.27 % (Jacob and Sreejith, 2008) although any other false alarm probability can be used. The actual acceptable false alarm probability should in fact depend on the actual product or process (Gokhale and Trivedi, 1998). Both types of errors are typically associated with economic loss.

**Parameter Estimation**

Parameter estimation is of primary importance in software reliability estimation. The parameter estimation methods can be of two types 1) Point estimation 2) Interval estimation (Pham, 2006). Two most popular estimation techniques are MLE and Least Squares Estimation (LSE). The MLE technique estimates parameters by solving a set of simultaneous equations. It is the most widely used estimation technique. In many cases, the maximum likelihood estimators are consistent and asymptotically normally distributed as the sample size increases (Zhao and Xie, 1996). In this thesis, the parameters are estimated by MLE technique for Time domain data.

**Maximum Likelihood Estimation Method**

The method of MLE is one of the most useful techniques for deriving point estimators. The idea behind maximum likelihood parameter estimation is to determine the parameters that maximize the probability of the sample data. A MLE method is versatile and applies to many models and to different types of data. Although the methodology for maximum likelihood estimation is simple, the implementation is mathematically intense.

If we conduct an experiment and obtain \( N \) independent observations, \( T = t_1, t_2, \ldots, t_N \). The probability density function of ‘\( k \)’ unknown parameters \( \theta_1, \theta_2, \ldots, \theta_k \) is given as \( f(t; \theta_1, \theta_2, \ldots, \theta_k) \). Assuming that the random variables are
independent, then the likelihood function, \( L(T; \theta_1, \theta_2, ..., \theta_k) \), is the product of the probability density function evaluated at each sample point:

\[
L(t_1, t_2, ..., t_N | \theta_1, \theta_2, ..., \theta_k) = L = \prod_{i=1}^{N} f(t_i; \theta_1, \theta_2, ..., \theta_k)
\]

The maximum likelihood estimator \( \hat{\theta} \) is found by maximizing \( L(T; \theta_1, \theta_2, ..., \theta_k) \) with respect to \( \theta \). The symbol (^) is used here to distinguish Maximum Likelihood estimators from the parameters being used.

Likelihood function by using \( \lambda(t) \) is expressed as,

\[
L = e^{-m(t)} \prod_{i=1}^{n} \lambda(t_i)
\]

Taking the natural logarithm of the above equation, we can obtain \( \ln L \).

The logarithmic likelihood function is given by,

\[
\log L = \log \left( e^{-m(t)} \prod_{i=1}^{n} \lambda(t_i) \right)
\]

The ML estimates (Cohen, 1965) of the unknown parameters \( \theta_1, \theta_2, ..., \theta_k \) are obtained by maximizing and differentiating \( \ln L \) with respect to each of the unknown parameters. Solving the equations simultaneously and equating to zero.

\[
\frac{\partial(\log L)}{\partial \theta_j} = 0; \quad j = 1, 2, ..., k
\]

1.4 Sequential Probability Ratio Test (SPRT)

Sequential analysis was first developed by Abraham Wald in the 1940’s and (Wald, 1945; Wald and Wolfowitz, 1948) and introduced the SPRT during that time. The Sequential Probability Ratio Test (SPRT) was initially developed by Wald (1947) at Columbia University 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. A big advantage of
sequential tests is that they require fewer observations (time) on the average than
fixed sample size tests. SPRTs are widely used for statistical quality control in
manufacturing processes. 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)

Regarding estimation / prediction of software reliability through model building, our
interest got inclined to the work of Stieber(1997) that deal with detection of
unreliable software components using the theory of Wald’s Sequential Probability
Ratio Test (SPRT) procedure. We made an attempt to adopt the SPRT methodology
to our proposed Burr Type XII model for Time domain data to detect whether the
software product is reliable / unreliable (Stieber 1997).

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). The time-domain approach involves recording the
individual times at which failure occurred. The interval-domain approach is
characterized by counting the number of failures occurring during a fixed period
(e.g., hour, week, day). These data are usually used by practitioners when analyzing,
assessing and predicting reliability applications. In this thesis the software reliability
growth model handles only Time domain data (i.e., Ungrouped).

1.5 Burr Type XII Software Reliability Growth Model

There are essentially two types of software reliability models-those that attempt to
predict software reliability from design parameters and those that attempt to predict
software reliability from test data. The first type of models are usually called “defect
density” models and use code characteristics such as lines of code, nesting of loops,
external references, input/outputs, and so forth to estimate the number of defects in
the software. The second type of model is usually called “software reliability growth
models”.
**Burr Type-XII SRGM:** The mean value function of Burr type XII is given by

\[ m(t) = a \left[ 1 - (1 + t^c)^{-b} \right] \]

The unknown parameters ‘a’, ‘b’ and ‘c’ are estimated by using New-Raphson method.