CHAPTER 1
Monitoring Time Domain Based Exponential Software Reliability using SPC

1.1. INTRODUCTION

Today Society is increasingly dependent on products and services based on software, i.e. large-scale systems such as health, transport, banking, telecommunication etc, to household products such as Televisions, Refrigerator, Washing machine, Air Conditioners, mobile phones etc., Organizations need to develop reliable software with high quality, demanded by the customer. Methods that allow for better resource planning and scheduling, in addition to cost reduction by effective verification, are therefore highly desirable.

Techniques that enable developers to improve reliability before the system is taken into operation are beneficial. Identification of factors affecting software quality can facilitate development of such methods. Knowledge about which development factors have a significant impact on the project, can also guide process improvement efforts. The identity and impact of factors affecting the software quality in a given organization are often concealed in the complexity of software development and therefore strategies to define and validate methods with more useful guidelines are required. Both qualitative and quantitative research results can facilitate the identification of contributing factors. 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 can form the basis for identifying the process variation using software defect data, thereby, improving software quality and reliability.
1.2. Software Reliability

Software reliability is defined as “The ability of a system or component to perform its required functions under stated conditions for a specified period of time.”

Software reliability 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.

Faults especially with logic, in software design are thus more subtle. 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. Software development cost and the penalty cost of software failures has become a major expensive in the whole system (Pham 1992a). Failures of the software may result in an unintelligent 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. Studies conducted by Micro Soft showed that it takes about 12 programming hours to locate and correct a software defect. At this rate, it can take more than 24000 hours to debug a program of 35,000 Lines of Code at a cost of over US$1 million.

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.

Software reliability 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.
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, 200a). 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 base on Non Homogeneous Poisson Process (NHPP) Models.

1.3. 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. The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs [BS 4778: Part 1: 1987 (ISO 8402:1986)]

The total composite product and service characteristics of marketing, engineering, manufacture, and maintenance through which the product and service in use will meet the expectation of the customer (Feigenbaum).

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.

Since software is an intellectual creation, it is difficult generally to measure the software quality. However, it can be judged by its correctness, adaptability, testability, obsolescence and intra-operatability. In addition to this a software product has a typical development process as some activities are related to particular phases of development and the other activities of the software apply throughout the software development. As in the case of any other product the only way for a software quality is to identify systematically the opportunities of improvement at every stage of software development.
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 whereby 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.

1.4. 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. (Johan S Okland, 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 instance of 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 and off 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. Because these requirements vary for each customer and often times conflict, even whining about a given instance of the software process.

Despite the difficulties mentioned, it may be still possible to apply SPC to software process. First of all, the process that needs statistical control should be identified.

1.5. Statistical Process Control
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 (Sutherland et. al., 1992). The basic tools used for statistical control are (Paulket. al., 2002, Montgomery, D.C, 1991)

1. Check Sheet
2. Cause-and-Effect Diagram
3. Scatter Diagram
4. Run Chart
5. Histogram
6. Bar Chart
7. Pareto Chart
8. Control Chart

1.5.1. Check Sheet
Check sheets are good means for collecting data efficiently, reliably and easily. As the detail and characteristics of data are different, check sheets are designed specifically considering the particular needs. In software industry, metric datasheet is in form of a check sheet, which is used to collect measurement data. Although automated measurement techniques minimize the human interaction for data collection, metric datasheets are still used extensively in order to represent the data in the desired format.

Table 1.1: Sample Defect Density Table
1.5.2. **Cause-and-Effect Diagram**

Cause-and-Effect Diagrams are useful tools to visualize, categorize and rank potential causes of a problem, a situation or any outcome. They are also named as fishbone diagrams because of their shapes and are usually formed as a result of a discussion or a brainstorming session of a group of people.

![Sample Cause-and-Effect Diagram](image)

**Figure1.1: Sample Scatter Diagram**

1.5.3. **Scatter Diagram**

In order to draw a Scatter Diagram, data for two variables are collected in pairs \((x_i, y_i)\), and each point \(y_i\) is plotted against corresponding \(x_i\). This is a useful plot for identifying a potential relationship between two process
characteristics. A pattern in the plotted points may suggest that the two factors are associated, perhaps with a cause-effect relationship. Moreover, scatter diagrams may be used for regression analysis if the necessary assumptions are satisfied.

![Scatter Diagram](https://via.placeholder.com/150)

**Figure 1.2: Sample Scatter Diagram**

1.5.4. **Run Chart**

Run Charts are specialized, time-sequenced form of scatter diagrams that can be used to examine data quickly and informally for trends or other patterns that occur over time. They dynamically observe performance of one or more processes over time. They look like control charts, but without the control limits and center line. They are useful for visualizing performance after a process change. In Run Charts, there should be at least 20-25 points and y axis should be 1 ½ times the range expected (Radice, R, 1998)

![Run Chart](https://via.placeholder.com/150)

**Figure 1.3: Sample Run Chart**

1.5.5. **Histogram**

Histograms show the frequency distribution of data in a sample. The first step to draw a histogram is to categorize the data into classes with equal
ranges. Data sets with a large number of elements usually require a larger number of classes, whereas smaller data sets require fewer classes. As a rule of thumb, the number of classes should be between 5 and 15. Then the number of data in each cell is found and depicted with bars on the graph. The data represents the state of a system at a certain time; thus there is no time dimension. These charts are quite practical to visualize central tendency and skewness of the measured attribute.

**Figure 1.4: Sample Histogram**

### 1.5.6. Bar Chart

Bar charts are like histograms. But they are not only used for depicting the frequencies of occurrences, but also for showing any numerical value of the attribute.

**Figure 1.5: Sample bar Chart**
1.5.7. Pareto Chart

Pareto chart is another form of bar chart. However, the occurrences are ordered with respect to their frequencies. Pareto charts are good means to visualize the ranking of an attribute among different categories.

![Sample Pareto Chart](image)

**Figure 1.6: Sample Pareto Chart**

1.5.8. Control Chart

Control Charts are sophisticated statistical data analysis tools, which include upper and lower limits to detect any outliers. They are frequently used in SPC analyses and will be described in detail in the following sections.

![Sample Control Chart](image)

**Figure 1.7: Sample Control Chart**
1.6. Variability in Processes

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 (Shewhart, W.A, 1939). 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, source lines of code produced a day can be considered as a variable parameter. If the same person produces SLOC 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, W.E, 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.

1.6.1. Statistical Control

*Control* means predictability. 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.

1.6.2. Shewhart Control Charts

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 statistics and control charts (Shewhart, W.A, 1939). Shewhart control chart model depends on hypothesis testing. First of all, a sample of data is collected for the subject measure (i.e. number of defects in a piece of code). Then, its mean and variance are calculated. 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, whereas 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. (Exponentially weighted moving average charts and CUSUM)
1.6.3. Reliability and Control Charts

The control chart is considered as the formal beginning of the statistical quality control. The Control chart is one of the seven (often referred to as the magnificent seven) tools of Statistical Process Control (SPC). Statistical process control (SPC) can be defined as a collection of tools, which track the statistical behavior of production processes, in order to maintain and improve product quality, thereby reliability. The ideology behind SPC is similar to that of other quality and reliability philosophies like Total Quality Management (TQM) and Six Sigma. Therefore, SPC is regarded as an important component of Total Quality Management (Cheng and Dawson, 1998) and other quality philosophies. Of all the tools of Statistical Process Control, Control charts are, perhaps, most technically sophisticated. The basic idea behind any control chart is to monitor a process and to identify any unusual causes (also referred to as assignable causes) of variation from the chance causes of variation (inherent to the process) to assess reliability.

1.6.4. Utilization of SPC in Software

The interest to apply SPC techniques in the software industry has been growing during the last decade as more organizations advance in maturity levels of process improvement models such as Capability Maturity Model (CMM) (Paulk, M. C, 1993), Capability Maturity Model Integration (CMMI, 2001) and SPICE (ISO/IEC TR 15504-5, 1998) These models implicitly direct software companies to implement SPC as a crucial step for achieving higher process maturity levels. They suggest control charts for both project level process control and organizational level process improvement purposes. In the literature, there are several resources on the usage of statistical techniques in software development (Card, D, 1994), (Kan, S. H., 1995), (Lantzy, M.A., 1992), Some researchers contribute to this trend by providing approaches to utilize SPC techniques for software industry Moreover, most of the examples exhibited in the studies refer to defect and inspection data (Paulk, M. C., Chrissis, M. B., 2002).
1.7. Research Literature and Proposed Study

Trivedi and Shooman (1975) suggested a many state Markov model for the purpose of estimation and prediction of software performance parameters. The availability and reliability of the software system are also presented for the purpose of predictions.

Based on the notion of the distribution function of random variable Forman and Singpurwalla (1978) developed a probabilistic model describing the software failure phenomenon to suggest estimates of the parameters in the model and termination procedure for debugging the software.

Schick and Wolverton (1978) describe the most commonly used software reliability growth models as divided into two groups of time domain and data domain. They present a comparative picture of the model as applicable to various actual data sets.

Goel and Okumoto (1979) are two of the foremost researchers that have considered the probabilistic nature of software failure phenomenon based on an NHPP. They have analyzed the failure process to develop a suitable mean value function which in turn is used to get software performance measures like distribution of the cumulative number of software errors, the residual number of errors, software reliability, maximum likelihood estimators of the parameters of the model and joint confidence region of the parameters. The mean value function of this model turns out to be based on the cumulative distribution function of the well-known exponential distribution.

Musa (1980) presented the need for potential use of software reliability measurement and made a comparison of software and hardware reliabilities. The concepts of software reliability, software and hardware components availability are also presented. Ramamurthy and Bastani (1982) reviewed the status and perspectives of software reliability as on 1982.

Iannino et al., (1984) gave a descriptive narration of various criteria for the comparison of software reliability models based on predictive validity, capability, quality of assumptions, applicability, and simplicity.

Kazuhira Okumoto (1985) proposed a statistical method that can be used to monitor, control and predict the quality of a software system being tested. The method
consists of three steps: estimation of the failure, intensity based on groups of failures, fitting the logarithmic poison model to the estimated failure intensity data, and constructing confidence limits for the failure intensity process. Amrit L. Goel (1982) presented an overview of the key modelling approaches provided a critical analysis of the underlying assumptions, and assessed the limitations and applicability of software reliability models during the software development life cycle. Singpurwalla and Soyer (1985) suggested random coefficients auto regressive process to assess software reliability growth and applied their approach to a real life time data.

Matsumoto et al.,(1988) discuss the evaluation procedure of a SRGM using data from a single program testing process applied to exponential, hyper exponential and S-shaped models, ranking the S-shaped model as superior with respect to estimation as well as goodness of fit.


Humphrey, W (1989) can be regarded as a reflection of quality management on softwareengineering discipline. He describes a framework for software process management, outlines the actions to provide higher maturity levels and acts as a basic guide to improve processes in a software organization. In this book, Statistical Process Control appears as a means of data analysis technique for level 4 organizations. Humphrey emphasizes that measures should be robust, suggest an norm, relate to specific product and process properties, suggest an improvement strategy and be a natural result of the process. He also mentions that it is essential to have a model, but believing it too implicitly can be a mistake. As SPC is more or more regarded in software industry, additional studies are being performed by the researchers. W.Steven Deummy, Arthur B.Petrini (1989) shows how SPC technologies may be used to improve the quality and productivity of large-scale software development. They also described the steps required to apply SPC to software development activities and identified activities that appear to be particularly good SPC candidates. And also summarized major advantages and disadvantages of the SPC approach to software development.
Fault density and failure intensity of an SRGM are used as two metrics to monitor software development capabilities and to measure customer satisfaction with the developed product in the research investigation of Huensch et al. (1990). Ehrlich et al. (1990) used the software reliability data collected during the testing of a system to measure the software quality in terms of experienced software failures. The notions of exponential NHPP are made use of for goodness of fit of the data. Csenki (1990) presented the use of Bayes prediction analysis to derive the predictive distribution of time to next failure. The concept is applied to the Jalenski-Moranda software reliability model. The software reliability growth, decay in a given period of software failure data specifically a software production concern is evaluated by Bastos et. al. (1990) using Laplace trend analysis for exponential and S-Shaped growth models. Jacoby and Tohma (1991) used hyper geometric distribution to estimate the number of initial faults in a software at the beginning of the test and debug phase. The time to repair, mean logistic delay time variants at the system level are modeled by Crow (1990).

Tohma et. al., (1991) investigated six ways of the estimation of parameters in a hyper geometric distribution to get an estimate of the number of initial faults in a program at the beginning of its testing/debugging, along with their relative accuracies. Sofer and Miller (1991) use a nonparametric method of estimating the software failure rate in completely monotone models which can be compared with parametric approaches. Shooman (1991) discusses a micro software reliability model that can be used to apportion of reliability and test efforts among the various execution paths. Vallee and Ragot (1991) demonstrated the application of NHPP approach to the industrial world generating accurate predictions with specific applications in space research.

Lyu and Nikora (1991a) proposed linear combinations of software reliability models for the purpose of automating the procedures of software reliability analysis. The concept of equally weighted linear combination model resulting from linear combination of three popular software reliability models is suggested by Lyu and Nikora (1991a) for applying to software failure data sets. Tohma et. al., (1991) made use of hyper geometric distribution to estimate by least squares sum method the number of faults initially present in a program at the beginning of test/debug process arriving at reduction in the time needed for calculating the estimates.
Jacoby and Tohma (1991) explained the idea of capture-recapture process for software faults in the context of a proposed testing environment. Estimation of the degree of unavailability of the software system is also given for hyper geometric distribution.

Mark A. Lantzy (1992) proposed SPC based solutions which can be applied, raging from the creation of the informal environments of a garage shop to the implementation of the formalized processes of software factories. Mark (1992) also mentioned that IBM Used SPC methods in the development of OS/2 version 2.0. He stressed the need for adopting SPC techniques as increasing demand for software continues to out-pace software production capabilities in terms of both quantity and quality.

Scott A. VanderWielet. al (1992) explained the goal of algorithmic statistical process control to reduce predictable quality variations.

Lantzy (1992) presents one of the earliest studies on the debate of applying Statistical Process Control to software processes. In this paper, he summarizes the concept of SPC and gives some practical examples from manufacturing industry. Then he offers a set of transformations on these principles via software quality characteristics revealing the uniqueness of software products. After giving the process-product relationship, he outlines a seven-step guideline for successful SPC implementation in a software organization. This study reveals four important points for the application of SPC to software processes:

- Metrics should correlate to the quality characteristics of the products that are defined by the customer
- Metrics should be selected for the activities that produce tangible items
- SPC should be applied only to critical processes
- The processes should be capable of producing the desired software product

If testing and operation are the two phases of a developed software, taking into considerations the notion of software reliability during testing phase an SRGM is suggested by Yamada et al., (1993). This model incorporates the amount of test effort as a random variable following a Weibull distribution. An SRGM useful for operational use is suggested by Kenney (1993) that can estimate the residual
number of defects, anticipated arrival times of customer reported failures taking into
the concept of “power function of time” associated with Weibull distribution. The
concept of data aging is made use of to identify the starting failure count of a
software failure data in an optimum way to choose an SRGM for failure data
analysis by Schneidewind (1993). This model suggests four optimal criteria to
determine the start of a failure count. Necessary and sufficient conditions for the
existence and finiteness of the MLEs of the parameters of SRGMs are derived by
Hossain and Dahiya (1993) and applied to various standard models.

Confidence interval procedures for power law based on NHPP SRGM are
suggested by Crow (1993). Gaudoin et. al., (1994) generalized the concept of
quantified quality debugging of Gaudoin and Soler (1992) from deterministic to
probabilistic and suggested another model called log normal proportional model as
software reliability model. William T. TUCKER and Frederick W Faltin (1993)
presented a scheme for uniting traditional SPC and feed forward/ feedback control
into a system that exploits the strengths of both. Hossain A. Syed et. al (1993)
presented a necessary and sufficient condition for the likelihood estimates to be
finite, positive and unique and suggested a modification to Goel and Okumoto
model.

Based on the concept of different rates of debugging at the different stages of a
software development, Kececioglu et. al., (1994) worked out a reliability growth
model using the well known Gompertz curve which is a special fit for failure data
with S-shaped trends. Sylvia and Singpurwalla (1994) propose a Bayesian approach
for predicting the number of failures in a software using the log Poisson execution
time model based on published information on the empirical experiences of other
researchers. They have named such an information as expert opinion. Whittaker and
Thamson (1998) described a method for statistical testing based on a Markov chain
of model of software usage with a two-fold significance. The influence of failure is
assessed and a stoppage rule for the testing process is derived.

The two main characteristics of a software development process are:

1. No programmer is perfect and thus when an error is removed new errors can
be introduced into the program.
2. Not all errors are created equal. Different errors will have different implications and need different handling.

The first characteristic mentioned above is called imperfect debugging. Some people who worked in these areas are Kuo (1983), Yamada et al., (1984), Yamada and Osaki (1985, 86), Ohoba and Chou (1989), Kareer et al., (1990), Leung (1992), Kapur and Bhalla (1992). Combining the above two characteristics Lynch et al., (1994) developed a software reliability model to determine the optimal debugging time necessary to minimize the cost without disturbing reliability constraints. Kapur et. al., (1994) incorporated the concept of imperfect debugging with testing effort in a software reliability model that can be used to plan the amount of testing effort required to achieve a predetermined target in terms of errors removed in a given span of time.

As Card (1994) points out, SPC and other measurement based techniques are based on the assumption that the organization has defined processes. Thus, before starting to work on the application of SPC techniques, it is essential to have an understanding about the measurement processes and metrics. Fortunately, software area has extensive resources on measurement processes.

Card (1994) discusses the utilization of SPC for software by also considering some of the objections and mentioning possible implementation problems. He gives an example control chart for testing effectiveness measure and concludes that SPC principles can be beneficial for a software organization although formal statistical control techniques may not be used.

Kan (1995) provides detailed information about software metrics, software reliability models, and models and analysis of program complexity. The book also includes examples from real-life practices and presents empirical knowledge to the software community. The author emphasizes the difficulty in achieving process capability in software domain and is cautious about SPC implementation. He mentions that the use of control charts can be helpful for a software organization especially as a supplementary tool to quality engineering models such as defect models and reliability models. However, it is not possible to provide control as in manufacturing since the parameters being charted are usually in-process measures instead of representing the final product quality. The final product quality can only be measured at the end of a project as opposed to the production
in manufacturing industry, so that on-time control on processes becomes impossible. He also underlines the necessity of maturity for achieving process stability in software development. Finally, he brings a relaxed understanding by stating that the processes can be regarded in control when the project meets in-process targets and achieves end-product quality goals.

Burr and Owen (1996) describe the statistical techniques currently available for managing and controlling the quality of software during specification, design, production and maintenance. This book is one of the very few resources in the area as it is a full reference on statistical methods from technical background of statistics and measurement to managerial concerns in software industry. The main focus is given to control charts as beneficial SPC tools and guidelines are provided for measurement, process improvement and process management within the software domain.

The theory of order statistics of statistical inference is made use of by Mitchell and Zeil (1996) in presenting a software reliability model with a notion of directed testing method.

Software reliability prediction in a nonparametric scenario through Bayesian approach is studied by Aroui and Soler (1996). The optimal testing rule and hence the optimal release policy for a model based on hyper geometric distribution are investigated by Hou et al. (1996).

Shima et al. (1997) examined the failure intensity distribution of an SRGM from an empirical failure data and show that the procedure predicts a number of detected faults more accurately than the conventional ones.

The software aging phenomenon is an equally important notion that has to be given some consideration. This is expressed in terms of accumulated errors during the execution of the software. The effect of aging is quantified with the help of a metric called estimated time to exhortation. These aspects are studied in Garg et al. (1997).

The Bayes estimate of mean number of errors present in a software system, the system reliability along with numerical illustrations are given by Kuo et al. (1997). Safety critical analysis applied in reliability modeling in order to increase the trust in the reliability of such products is the contribution of Schneidewind (1997) by
integrating software safety criteria, risk analysis, reliability prediction and stopping rule for testing. Cai (1997) considered the non failure stops of software execution process as a type of censored data and developed a reliability model by censoring the existing software reliability models. Pointing out the inadequacy of some SRGMs in describing the failure process of a software failure data, Fenton (1997) defines measurement as ‘the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules’. Here an entity stands for a process or a product whereas an attribute is a feature of the entity. Fenton and Pfleeger (1997) provide valuable information by considering the whole life cycle of measurement from planning to data analysis in a software organization. They also define the basics of some mostly used metrics and provide empirical information in software organizations.

Incorporating the concept of explicit debugging activities along with the possibility of imperfect debugging, Gokhale et al., (1998) have suggested software reliability analysis with fault detection and debugging. The study is based on the estimates of residual number of faults in the software. Gokhale and Trivedi (1998) propose the mean value function of an NHPP as given by a log logistic model that can indicate the increasing as well as decreasing nature of the failure occurrence.

Software reliability modeling through concatenating failure rates to overcome the lack of memory feature of a Poisson process is suggested by Singpurwala (1998). Thomson and Whittaker (1998) considered the combination of the notions - software failure as rare event and finite state discrete parameter recurrent Markov chain. The results are applied to the analysis of software reliability for systems of high quality. Radice (1998) describes SPC techniques constrained within software domain and gives a detailed tutorial by supporting his theoretical knowledge with practical experiences. He states that all SPC techniques may not be applicable for software processes and gives XmR and u charts as possible techniques. He also explains the relevance of SPC for CMM Level 4 and regards back-off of control charts in Level 4 as a mistake. He states five problems with control charts: too much variation; unnecessary use of control charts; lack of enough data; lack of specification limits from the clients; the idea that control charts cannot be used with software processes.

Florac and Carleton (1999) represent CMM understanding on the utilization of Statistical Process Control for software process improvement. The book includes
detailed technical information about SPC and provides a roadmap for SPC implementation. It mostly focuses on the benefits of control charts depending on Shewhart’s (1939) principles. It also discusses some issues related to the application of control charts in software development and incorporates the experience gained from manufacturing industries to software processes. Evanco (1999) used the notion of proportional hazard model in the form of a software development by identifying the three explanatory variables namely software complexity, software development volatility and cumulative execution time as variables influencing software reliability. He developed the regression approach to effect the reliability of a software by these three variables. Relaxing the nature of statistical independence that is generally assumed among successive software failures, Goseva and Trivedi (2000) suggested software reliability modeling frame work based on Markov renewal process. This modeling becomes a general case of many models that assume failure independence. Keiller and Mazzuchi (2000) investigated the improvement in the performance of Goel-Okumoto model with various smoothing techniques including Laplace trend test.

Weller (2000) provides a distinct study by presenting the SPC implementation details from a software organization. In order to regard defect density as an indicator of product quality, he first wants to be sure that inspection process is stable in the organization. He uses X and moving range charts for the lines of code inspected per hour for each inspection and achieves a stable inspection process after removing the outliers from the dataset. Then he draws u-chart for the defect density data for each inspection. By using these findings, he makes reliable estimations for inspection effectiveness and gains an insight on when to stop testing. This study carries the idea of SPC beyond theory and shows its benefits in providing control of process data and in creating a performance base for making predictions in real application environment.

Pham and Pham (2001) studied the predictive performance of a software reliability model inserting pseudo failures based on Bayes approach. Combination of failure detection and fault correction in a single model is done by Schneidewind (2002) in order to have a prediction support. Reliability assessment of software divided into a finite number of models with the help of a Markovian model and that of statistical test of hypothesis, is studied by Rajgopal and Mazumdar (2002). Homogenizing failure data with respect to failure intensities and adopting different SRGMs for
different failure intensities, a combined SRGM for a better reliability assessment is suggested by Tian (2002). Keiller and Mazzuchi (2002) studied the performance of a set of SRGMs using smoothing techniques including Laplace trend test.

Jakolte and Saxena (2002) move ahead on the idea of 3 sigma control limits and propose a model for the calculation of control limits to minimize the cost of type 1 and type 2 errors. The foundation of this study is a pioneering one as it questions an accepted practice for control charts and the results of the example studies are encouraging. However, the study is more academic rather than a practical one as it includes too many parameters and assumptions.

Huang and Kuo (2003) describe a unified scheme of estimation in an SRGM using weighted arithmetic mean, weighted geometric mean and weighted harmonic mean. Their unified approach was verified to cover many of the well known NHPP models under different sets of conditions. Boland and Singh (2003) adopted a birth process approach to Moranda geometric software reliability model and derived the mean value function, intensity function, reliability function, probability distribution of relevant point process and optimum release time for the software. A concept similar to imperfect debugging is fault removal efficiency. Combining the notions of fault removal efficiency, failure rate and fault introduction rate a unified model that integrates these three is proposed by Zhang et al., (2003). On the assumption of improvement in reliability along with time irrespective of bugs are corrected or not, a simple software reliability model is proposed by Jalote and Murphy (2004).

The notions of faults dependency and time dependent delay are introduced by Huang et al., (2004) and described an SRGM with imperfect debugging. A unified mathematical modeling for explaining the imperfect debugging in a software fault process by dividing the original failure process of the model into two different NHPPs is introduced by Huang (2004). Groen et. al., (2004) presented reliability data collection and analysis system – a software tool, in short- ReDCAS, employing Bayesian techniques to estimate reliability measures. The concept of infinite time to the next failure is the general consideration of many SRGMs indicating that the software product is very good. Mullen and Gohale (2005) have considered a Poisson process with log normal distribution as the mean value function and presented goodness of fit tests for software failure data. The estimates of software failure rates adjusted in tune with the test data for the suitability of appropriate reliability models is considered by Jeske et. al., (2005).
An efficient software reliability assessment model for the actual open source system development to compare between S-shaped and other models is investigated by Tamura and Yamada (2005). Ohishi et al., (2005) proposed the NHPP based on Gompertz curve as a software reliability model and applied it to a live data. Keiller and Mazzuchi (2005) suggested some methods of comparing software reliability models using smoothing techniques. Crow (2005) gave a detailed note about the methods improving the effectiveness of the reliability tasks. The concept of multiple change points and imperfect debugging are integrated in a single model to study the software reliability by Huang and Lin (2005).

Hairulliza Mohammad Judi (2009) presented the implementation of quality control in three Malaysian companies and identified the factors that influence the selection of quality control techniques in these companies. And also identified that the initial role of SPC is to prevent product or process deterioration rather than identify product or process deterioration. Samuel E. Buttrey, (2009) presented a free add-in for excel that draws the most common sorts of control charts.

1.7.1. Proposed Study

We are motivated to study the following research problems related to MONITORING EXPONENTIAL SOFTWARE RELIABILITY USING SPC as these seem to be unattempted / unavailable in published form.

(i) We propose a control scheme based on the inter failure cumulative data between observations of failure. The said control scheme can be easily and fruitfully applied to monitor the software failure process for Exponential Imperfect Debugging based Non-Homogeneous Poisson Process (NHPP). Maximum Likelihood Estimation (MLE) is used for parameter estimation.

(ii) We also propose a control mechanism which involves evaluation of the parameter of the Mean Value function and hence the values of the mean value function at various inter failure times to develop relevant Mean.
value control chart. We want to adopt a modification for the likelihood function of first proposal, to set simple and efficient estimates.

(iii) We also propose a control mechanism based on Modified Maximum Likelihood of cumulative quantity between observations of time domain failure data using mean value function of exponential imperfect debugging based on NHPP.

(iv) We also propose a control mechanism based on order statistics of cumulative quantity between observations of time domain failure data using mean value function of Exponential Imperfect Debugging based on NHPP, if waiting for a fixed number of failures is acceptable.

(v) We compare our control mechanism with the control mechanisms developed using Half Logic Distribution and weibull distribution

The research findings out of our proposed study are presented with detailed discussion in the following chapters. Reprints of some of our results published in standard journals are appended towards the end of the thesis, with relevant reference in the introduction of the respective chapters.