APPLICATION OF STATISTICAL PROCESS CONTROL TO SOFTWARE DEVELOPMENT PROCESSES VIA CONTROL CHARTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

KAMİL UMUT SARGUT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

MAY 2003

Approval of the Graduate School of Informatics.

_____

Prof. Dr. Neşe YALABIK

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Semih BİLGEN

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Dr. Onur DEMİRÖRS

Supervisor

Examining Committee Members

Prof. Dr. Semih BİLGEN _____

Assoc. Prof. Dr. Nazife BAYKAL _____

Assoc. Prof. Dr. Onur DEMİRÖRS _____

Assoc. Prof. Dr. Ali DOĞRU _____

Dr. Altan KOÇYİĞİT _____

ABSTRACT

APPLICATION OF STATISTICAL PROCESS CONTROL TO SOFTWARE DEVELOPMENT PROCESSES VIA CONTROL CHARTS

Sargut, Kamil Umut

M.S., Department of Information Systems

Supervisor: Assoc. Prof. Dr. Onur Demirörs

May 2003, 124 pages

The application of Statistical Process Control (SPC) to software processes has been a challenging issue for software engineers and researchers. Although SPC is suggested for providing process control and achieving higher process maturity levels, there are very few resources that describe success stories, implementation details, and implemented guidelines for applying SPC to specific metrics.

In this thesis the findings of a case study that is performed for investigating the applicability of SPC to software metrics in an emergent CMM Level 3 software organization are presented. As being one of the basic and most sophisticated tools of SPC, control charts are used for the analysis. The difficulties in application of Statistical Process Control to a CMM Level 3 organization are observed by using the existing data of defect density, rework percentage, productivity and review performance metrics and relevant suggestions are provided for dealing with them.

Finally the analysis results are summarized and a guideline is prepared for software companies who want to utilize control charts by using their existing metric data.

Key Words: Statistical process control, control charts, software, CMM, case study, defect density, productivity, review performance, rework percentage.

ÖZET

İSTATİSTİKSEL SÜREÇ KONTROLÜNÜN KONTROL GRAFİKLERİ KULLANILARAK YAZILIM SÜREÇLERİNE UYGULANMASI

Sargut, Kamil Umut

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Onur Demirörs

Mayıs 2003, 124 sayfa

İstatistiksel Süreç Kontrolünün (İSK) yazılım süreçlerine uygulanması, yazılım mühendisleri ve araştırmacılar için önemli bir konu olmuştur. İstatistiksel Süreç Kontrolü, süreç kontrolünü sağlamak ve yüksek olgunluk seviyelerine ulaşmak için tavsiye ediliyor olsa da, belirli metrikler için başarı öykülerini, uygulama ayrıntılarını ve uygulanmış prosedürleri anlatan çok kısıtlı kaynak vardır.

Bu tezde, İSK'nın yazılım metrikleri için kullanılabilirliğini araştırmak amacıyla CMM seviye 3 olmuş bir yazılım firmasında yapılan durum çalışmasının sonuçları sunulmaktadır. Bu amaçla İSK'nın temel ve en gelişmiş araçlarından biri olan kontrol grafikleri üzerinde çalışıldı. Hata yoğunluğu, tekrarlanan iş yüzdesi, üretkenlik ve gözden geçirme performansı metriklerinin varolan verisiyle İSK uygularken karşılaşılan zorluklar gözlemlenip onlarla ilgili çözüm yolları önerildi. Son olarak analiz sonuçları özetlenerek kontrol grafiklerini kullanmak isteyen yazılım firmaları için bir yol haritası hazırlandı.

Anahtar Kelimeler: İstatistiksel süreç kontrolü, control grafikleri, yazılım, CMM, durum çalışması, hata yoğunluğu, üretkenlik, gözden geçirme performansı, tekrarlanan iş yüzdesi.

TO

Shewhart

ACKNOWLEDGEMENTS

I send my kindest regards and thanks to Mr. Demirörs for bringing about my affection to academic research and supporting endlessly during my graduate study.

I am grateful to İ.B., S.D., T.T.T., S.Y., S.N. and all other company personnel for their help and cooperation. And very special thanks to A.T. for her tremendous assistance, cheering sympathy and friendship during the case study.

I would like to express my sincere thanks to my family for always trusting me and tolerating all my fancies in the stressful days.

Most of all, I had the deepest emotions motivating me through long nights and days. Zeynep… I would never last during the hardest days in my life without you. Nothing to say, but thank you …

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

LIST OF ABBREVIATIONS

AIL: Action Item List

BMI: Backlog Management Index

CEO: Chief Executive Officer

CL: Control Limit

CMM: Capability Maturity Model

CMMI: Capability Maturity Model Integration

CoSQ: Cost of Software Quality

CPU: Central Process Unit

CSCI: Computer Software Configuration Item

CUSUM: Cumulative Sum

DCR: Document Change Request

FP: Function Point

IDD: Interface Design Description

IRS: Interface Requirements Specification

LCL: Lower Control Limit

NASA: National Aeronautics and Space Administration

PA: Process Attribute

PMBOK: Project Management Body of Knowledge

PR: Problem Report

QPM: Quantitative Process Management

SDD: Software Design Description

SLOC: Source Lines of Code

SOW: Statement of Work

SPC: Statistical Process Control

SPICE: Software Process Improvement and Capability Determination

SRS: Software Requirements Specification

TDCE: Total Defect Containment Effectiveness

UCL: Upper Control Limit

UITD: Unit Integration Test Description

UTD: Unit Test Description

CHAPTER 1

INTRODUCTION

The influence of software technologies on our daily life has grown exponentially during the last two decades. From a simple digital clock to telecommunication networks, software is extensively used to ease our way of living. Thus software can be regarded as "the gate to future" as it provides the basis for much of the technological advance in this century.

If we look at the history of software at a glance, we see that there is a rapid improvement in this sector since 1960s. However, software industry is quite young compared to manufacturing industries. After the industrial revolution, the manufacturing industry has come up to a level of stability, and the idea of continuous process improvement has been accepted throughout the world. In software, however, the improvement trend is too steep and organizations still strive for achieving a high level of maturity. Moreover, the characteristics of software make it complex and invisible, which make it difficult to apply the practices that are in use in other industries.

One of these common practices in manufacturing industry is statistical process control. The investigation on quantitative mechanisms as an aid to control process variation gave rise to the application of SPC since 1930s [1]. The idea of applying SPC to software development, however, is brought mainly by Capability Maturity Model (CMM) in mid 90s. Although its benefits are accepted for manufacturing companies, there have been many debates about its application in software development ([2], [3], [4]).

1.1. Difficulties and Benefits of SPC in Software Development

As Card [2] points out, SPC is founded on the principle that a process will demonstrate consistent results unless it is performed inconsistently. Thus, we can define control limits for a consistent process and check new process outputs in order to determine whether there is a discrepancy or not.

In the manufacturing arena, it is not difficult to figure out the relationship between product quality and the corresponding production process. Therefore we can measure process attributes, work on them, improve according to the results and produce high quality products. There is a repetitive fabrication of the same products in high numbers and this brings an opportunity to obtain high sample size for the measured attributes. Moreover, the product is concrete, and the attributes and variables to be measured are easily defined. Consequently, the only difficulty left is to define correct attributes and collect data for utilizing the tools of Statistical Process Control.

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. Crosby [5] defines quality as conformance to customer requirements. In addition, Lantzy [4] indicates that a software process must be assessed based on its capability to develop software that is consistent with user's requirements. 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 other 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 data 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. As the metrics are usually abstract, the interpretation of the data necessitates good judgment. Since the metric data are collected by regular project individuals a firm understanding of metrics should be provided for data validity.

Despite the mentioned difficulties, it may still be possible to apply Statistical Process Control to software processes. However, special consideration shall be given to the establishment of relevant mechanisms to gather beneficial outcomes. First of all, the key processes that need statistical control should be identified. Not every process needs statistical control, nor is related to organizational goals and process performance objectives. The variability may be inherently existent in the process, undermining the idea of using SPC. Moreover, it may not be economically and technically feasible to apply statistical control to some of the processes. SPC necessitates process stability for meaningful data analysis; thus the selected processes should have well-prepared process definitions.

Secondly, relevant measures for the selected processes should be defined. These measures should be informative about the desired process/product characteristics. For this reason, the relationship between the measure and the process/product it represents should be clearly understood. Moreover, it is critical to provide repeatability in measurement practices and consistency in the measured data. Therefore, relevant data collection mechanisms should be described in detail; and manual or automated means for data collection should be constructed. A database should also be created to accommodate the data as a baseline for the derivation of control limits within the organization.

Most of the times, the collected measure cannot be used directly in SPC analysis as some inherent parameters influence the generation of the data. Thus, it is essential to normalize the raw measures. These normalization procedures should be described in order to provide a consistent basis for meaningful comparison among different process instances. Meanwhile, relevant statistical techniques, control chart categories, and interpretation approaches should be identified.

Finally, these techniques should be utilized first to find and correct special causes, and then to visualize and improve processes.

If supported with reasonable data, necessary infrastructure and human resources, Statistical Process Control can be appropriately applied to gather beneficial information for software processes. First of all, the processes can be monitored with the establishment of process capability baseline and the process outcomes become predictable. Unpredicted behavior can be detected and necessary corrective actions can be implemented on time. As the process describes the way a product is produced, the quality of the product can be deduced from the related process and this enables the organization to produce high quality products by controlling the processes. After stability is achieved and special causes are removed, process improvement activities can be tracked through the analysis of common causes. The results of improvement actions can also be evaluated with the comparison of past and current conditions statistically.

1.2. 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) [6], Capability Maturity Model Integration (CMMI) [7] and SPICE ([8], [9]). 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 ([2] - [4], [6], [10] - [19]). Some researchers contribute to this trend by providing approaches to utilize SPC techniques for software industry ([10], [12] - [16]). Moreover, most of the examples ([10], [12] - [16], [17], [19], [20]) exhibited in the studies refer to defect and inspection data [21]. These studies however, focus on the potential benefits of SPC implementation, rather than providing a satisfactory guideline for software firms to implement SPC techniques with convincing information. We specifically lack

knowledge on the applicability of different metrics, the means of reliable data collection mechanisms, meaningful analysis approaches and practical evidence.

In this study, we investigated how SPC can be effectively applied to the processes of a software firm by using its existing measures. In order to visualize SPC implementation in real time, we performed a case study in an emergent CMM Level 3 software organization. After selecting a number of metrics, we performed a theoretical study to recognize metric basics and data analysis mechanisms. By working on the metric data, we established normalization procedures and translated the metric data to a form that is appropriate for comparison among different projects and products. As control chart is one of the most sophisticated data analysis tools within SPC, we demonstrated practical evidence on the utilization of SPC via control charts. Finally, we prepared a guideline for software companies in order to apply SPC techniques successfully and obtain the greatest benefits from an SPC implementation.

1.3. Roadmap

In Chapter 1, we give an introduction about our study and present the basics of our analysis. In Chapter 2, we establish an overall understanding on Statistical Process Control; we describe the methods used in statistical process control including check sheets, cause-and-effect diagrams, scatter diagrams, run charts, histograms, bar charts and pareto charts. Then we describe the concept of process variability and focus on control charts as a sophisticated SPC technique.

In Chapter 3, we investigate the extent to which SPC techniques are necessary and/or sufficient to conform to industrial software standards such as CMM, CMMI, SPICE, ISO 9000-3. We also highlight some of the software process guidelines and find out the clauses related to statistical methods. Then we cite several papers and books mentioning Statistical Process Control as an aid to visualize the evolution of SPC techniques in the industry. After summarizing the findings that inspired us during our study, we outline the difficulties related to the metrics that we worked on during the case study, make suggestions for proper metric analysis and describe means of using control charts.

Chapter 4 is devoted to our case study. We first give the reasons for selecting a case study and describe the fundamentals of our specific case. Then we focus on each metric separately and explain company-specific issues related to relevant software processes, measurement and data collection procedures. We also present details about our experiences while implementing SPC to metric data. Finally we summarize our analysis results.

In Chapter 5, we present our suggestions for a software firm as a guideline for achieving successful SPC implementation to the metrics that we handled during our case. In Chapter 6, we summarize our study, portray our overall findings and describe potential research opportunities on this subject.

CHAPTER 2

STUDY BACKGROUND

2.1. 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 [1]. The basic tools used for statistical control are ([21], [23]):

- Check Sheet
- Cause-and-Effect Diagram
- Scatter Diagram
- Run Chart
- Histogram
- Bar Chart
- Pareto Chart
- Control Chart

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. A sample metric datasheet is depicted in Figure 2.1.

Defect Density				
Project Name:				
Date	Software Component	SLOC	Number of Defects	Defect Density

Figure 2.1 Sample Metric Datasheet

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. A sample Cause-and-Effect Diagram can be seen in Figure 2.2.

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. A sample Scatter Diagram can be seen in Figure 2.3.

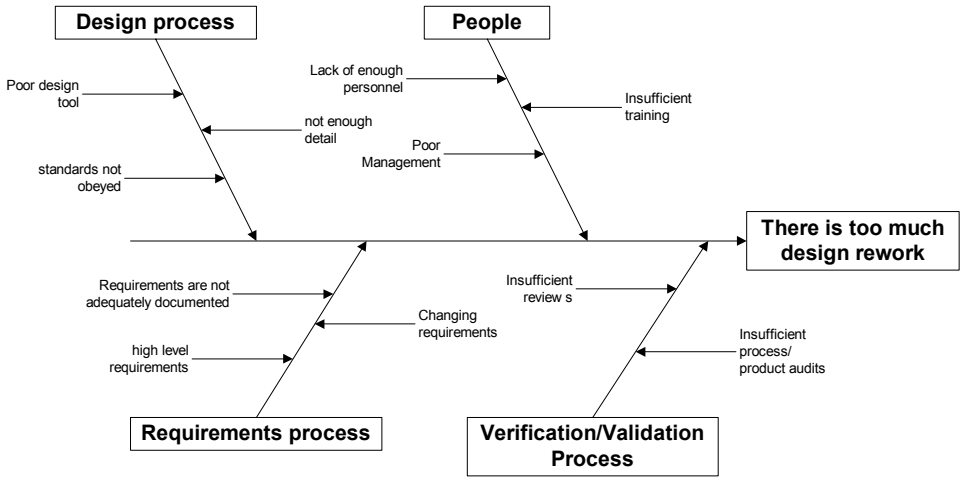

Figure 2.2 Sample Cause-and-Effect Diagram

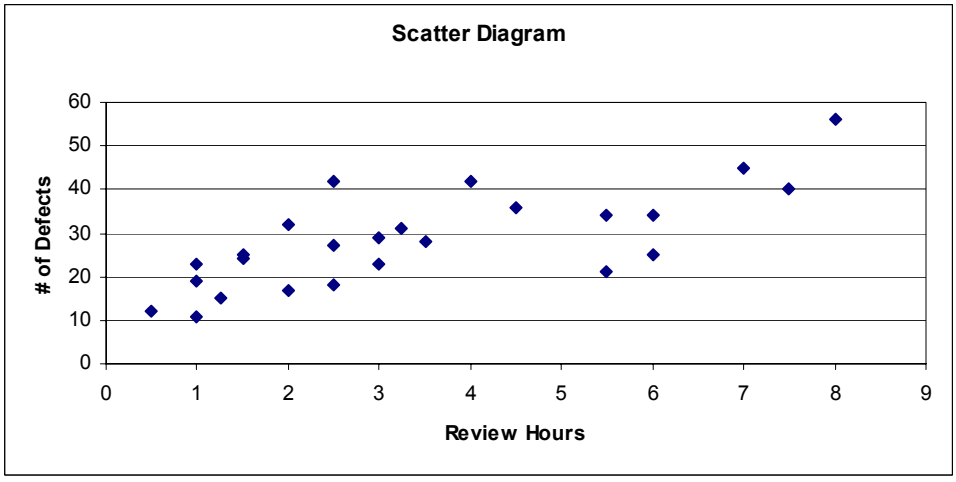

Figure 2.3 Sample Scatter Diagram

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 [17]. A sample Run Chart is given in Figure 2.4.

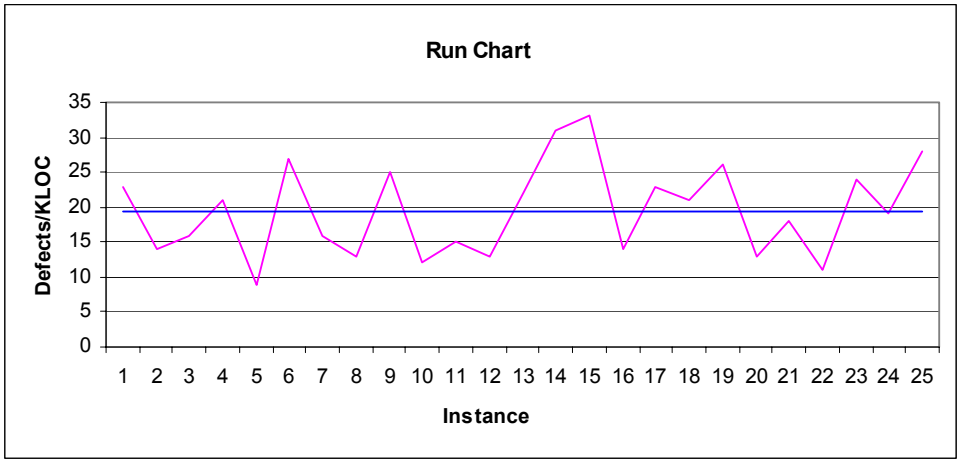

Figure 2.4 Sample Run chart

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 (see Figure 2.5). 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.

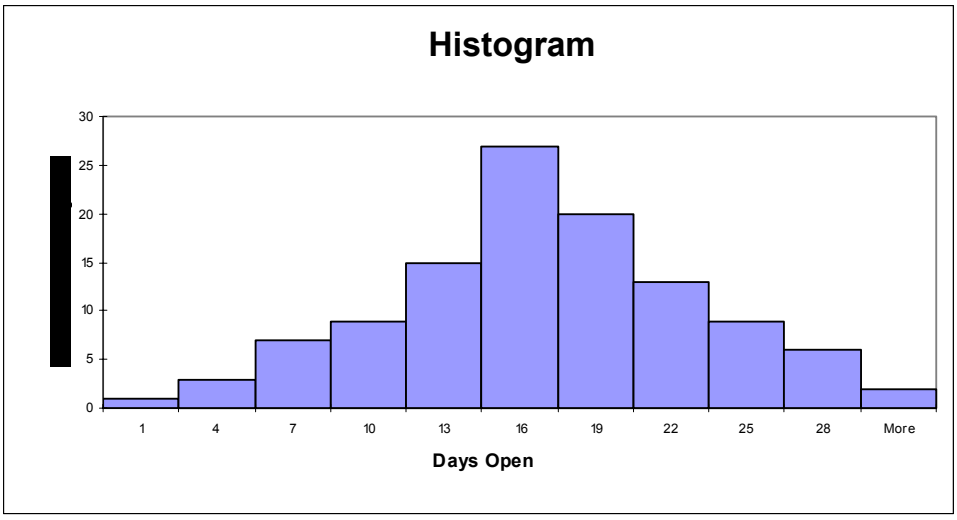

Figure 2.5 Sample Histogram

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. A sample bar chart can be seen in Figure 2.6.

Pareto Chart

Pareto chart is another form of bar chart. However, the occurrences are ordered with respect to their frequencies (see Figure 2.7). Pareto charts are good means to visualize the ranking of an attribute among different categories.

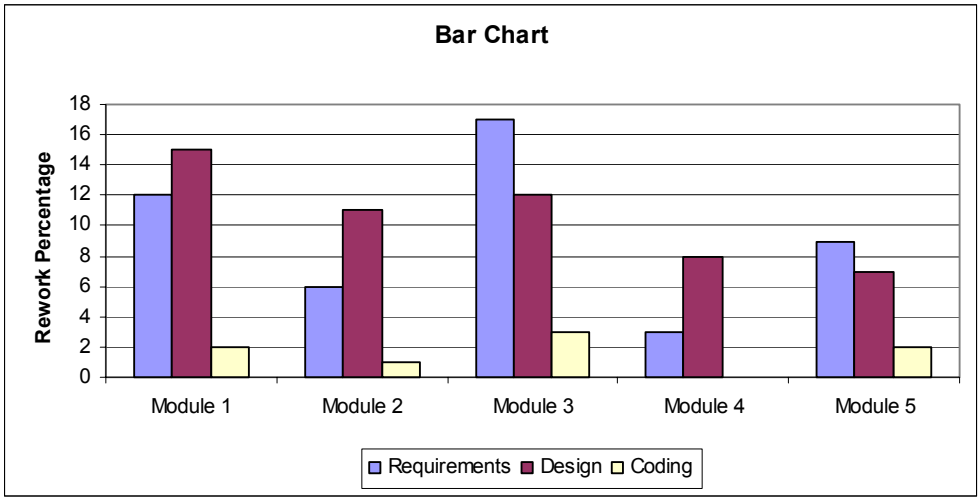

Figure 2.6 Sample Bar Chart

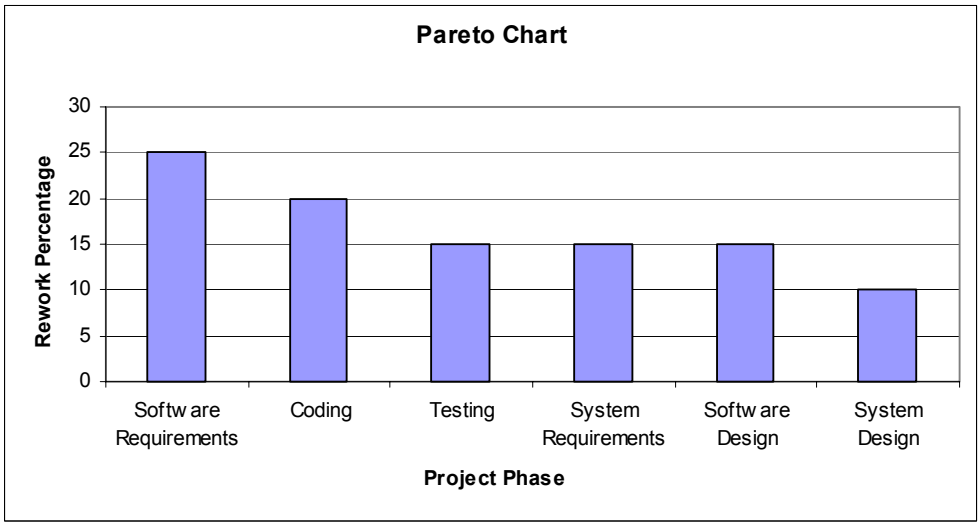

Figure 2.7 Sample Pareto Chart

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.

2.1.1 Concept of 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 [24]. 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. Either 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 [24], 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 [25] 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.

2.1.2 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 good means to achieve Statistical Process Control.

2.1.3 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 [24].

Shewhart control chart model depends on hypothesis testing. First of all, a sample of data (sufficient enough to represent the whole) 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. Burr and Owen [10] define 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*". 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 $\overline{X}$ 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 (cumulative sum) charts are also used for this aim.)

The limits for an Individuals Chart are calculated as:

$$\text{UCL} = \overline{X} + 3\,\frac{\overline{MR}}{d_2}$$

$$\text{LCL} = \overline{X} - 3\,\frac{\overline{MR}}{d_2}$$

where $\overline{X}$ is the average of individual values, $\overline{MR}$ is the average of moving ranges and d_2 is a constant which depends on the number of individual values that moving range is calculated [23]. If the moving range is calculated for 2 consecutive data values, d_2 takes the value of 1.128.

Moving range charts are also frequently used as complementary to Individuals Charts. For this reason, these charts together are referred as XmR Charts. As subsequent moving range values are correlated, patterns of cycles or runs are natural for the range charts. However, these charts are especially useful to view trends in the process.

The attribute measures are defined in sense of "presence or absence" (i.e. the number of defects) [1]. The fraction of defectives in a sample can be explained with a binomial distribution and p-chart is drawn to investigate statistical control.

The other major charts that may be utilized for the analysis of attribute data are the following:

***np charts*:** When we want to work on the number of defectives instead of the fraction.

***c-charts**:* If we want to emphasize the number of defects (of a certain type).

***u-charts**:* If we want to emphasize the number of defects per unit.

The limits for u-chart are calculated as:

$$\bar{u} = \frac{\sum d_i}{\sum n_i}$$

$$\text{UCL/LCL} = \bar{u} \pm 3\sqrt{\frac{\bar{u}}{n_i}}$$

where,

d_i: number of defects for measurement i

n_i: sample size for measure i

Shewhart charts are good indicators for detecting out-of-control situations in a process. For ease of understanding, the charts are divided into zones (A, B and C), which represent values of "Mean ± 1/2/3 Standard Deviation" as seen in Figure 2.8.

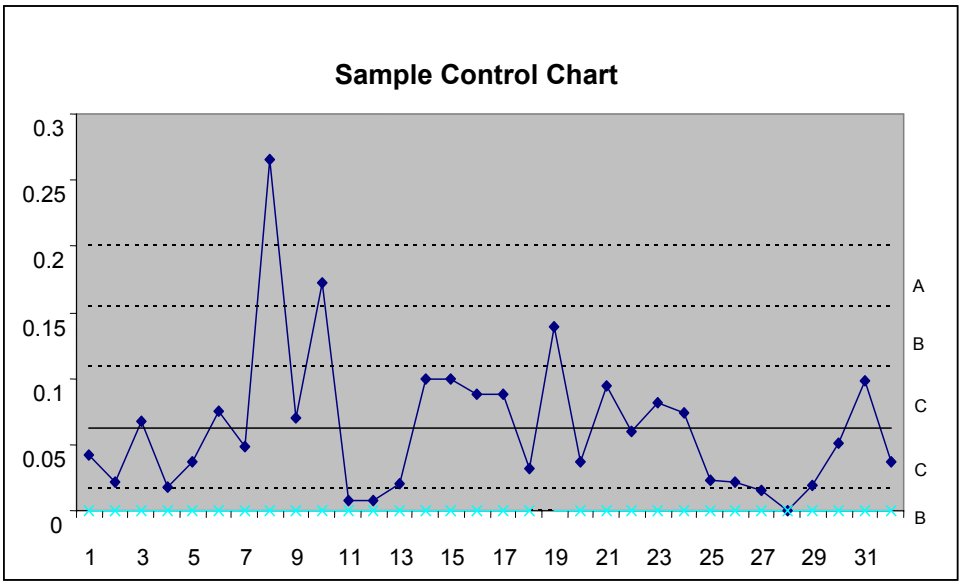

Figure 2.8 Sample Control Chart

Sutherland [1] has suggested the following test conditions for detection of violations in process control:

1) Extreme Points (Data beyond control limits)
2) Two out of Three points in zone A or beyond
3) Four out of five points in zone B or beyond
4) Eight or more runs above or below the mean
5) 6 successive points
6) 14 successive points oscillate up and down
7) 8 successive points not in Zone C
8) 15 successive points in zone C

These test conditions are applicable for different charts and give us evidence for deviation. By considering the nature of the process, the chart type, data

characteristics, current environmental conditions and other external factors, the process should be analyzed with the interpretation of the charts.

CHAPTER 3

RELATED LITERATURE

Human kind has always felt the need to evaluate tangible and intangible objects and processes. From the weight of sugar to the intensity of love (via love tests), we use measures in every aspect of our lives. Fenton [26] 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.

Software industry is not an exception and the intangible structure of software further increases the importance of measurement. For this reason, software measurement has been in concept of many authors, researchers and software engineers since late 1960s [5]. Today, statistical techniques begin to act as a supportive tool for the representation and analysis of measurement data with the enforcement of new trends. Moreover, the industrial standards and guidelines recommend utilization of different statistical techniques for successful process management. Among all the statistical techniques, the interest to apply SPC techniques in the software industry has also been growing during the last decade as more organization advance in maturity levels of process improvement models such as Capability Maturity Model (CMM) [6], Capability Maturity Model Integration (CMMI) [7] and SPICE ([8], [9]). 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. Many researchers contribute to this trend by providing approaches to utilize SPC techniques for software industry ([10] - [15]). Moreover, most of the examples exhibited in the studies refer to defect and inspection data ([10], [12] - [14], [17], [19], [20]). These studies however, focus on the potential benefits of SPC implementation, rather than providing a satisfactory guideline for software firms to implement SPC techniques with convincing practical evidence. We specifically lack knowledge on the applicability of different metrics, the means of reliable data collection mechanisms, meaningful analysis approaches and practical evidence. Nevertheless, the literature is very scarce in terms of practical experiences in application of statistical techniques in contrast to measurement.

In the following section we will investigate the place of statistical techniques in terms of necessity and sufficiency within industrial software standards such as CMM, CMMI, SPICE, ISO 9000-3. We will also go over some of the software process guidelines and try to make a connection between statistical methods and good software practices. In the second section, we will demonstrate our findings for the literature about the application of Statistical Process Control to software domain as an aid to visualize the evolution of SPC techniques in the industry. We will also mention about relevant citations that leaded us throughout our study with their content on measurement and SPC in general. Then we will summarize the current state of SPC techniques in the academic and industrial environments. Finally, we will go over each metric that we worked on during the case study and explain the fundamental issues regarding definition, analysis and interpretation of metrics.

3.1. Standards and Guidelines

A standard is a rule or basis for the comparison that is used to assess the size, content, value, or quality of an object or activity [20]. Thus, it defines what characteristics a product or a process shall possess. On the other hand, a guideline is a suggested practice, method, or procedure, typically issued by some authority [20]. Therefore, they lead process owners in performing their work although the rules are not mandatory. In practice, today's software standards include both

mandatory rules and suggested actions which are prepared by very experienced people in the related domain in long periods of time. Moreover, they evolve over time with the accumulation of new practices, tools, techniques and methodologies.

Because of the complexity of processes and the diversity of products, standards play a significant role for software development. And measurement is one of the important processes for which specific regulations are defined within software standards. The horizon of measurement is continuously growing day by day. Accordingly, the newer versions of software standards and guidelines begin to direct software organizations to utilize statistical techniques like reliability analysis, 6-sigma analysis and SPC.

Quantitative Process Management is one of the Level 4 key process areas of CMM [6], which involves establishing goals for the performance of the project's defined software process, taking measurements of the process performance, analyzing these measurements, and making adjustments to maintain process performance within acceptable limits. In the description of Commitment 1, the term "quantitative control" is referred to as any quantitative or statistically-based technique appropriate to analyze a software process, identify special causes of variations in the performance of the software process, and bring the performance of the software process within well-defined limits. Although the control chart is given only as an example for statistical analysis (not mandatory), CMM implicitly supports its utilization by mentioning about calculating mean and variances, identifying control limits and establishing process performance baselines. CMM also necessitates the collection of data from the projects to characterize the process capability in the organizational level. Therefore, the software organizations are trying to use control chart in order to meet the CMM Level 4 requirements.

On the other hand, CMMI [7] analyzes process performance in two process areas: Organizational Process Performance and Quantitative Project Management. The former mentions about establishing process performance baselines for the **organization's** standard software processes. The latter aims to quantitatively manage the **project's** defined process to achieve the project's established quality

and process performance objectives. Actually these two process areas work in harmony. The organizational process performance baselines, which are established by the data from all the projects, are used in order to control project level process performances. The project data are then added to the data pool in order to update organizational level process performance baselines.

QPM process area of CMMI aims to statistically manage a project's defined software processes (Sub-goal 2). CMMI differs from CMM in that it gives more detail about the statistical management. It gives more practical information to the software organizations as it is based on the realization that statistical techniques are not appropriate for all the processes. It necessitates identification of tasks and appropriate measures for those tasks to perform statistical analyses; it defines criteria for selecting the measures, and for determining whether historical data are comparable. Similar to CMM, control chart is given as an example for potential statistical analysis techniques. However, the requirements can best be met by implementing control chart analysis.

Although SPICE is an assessment model, it implicitly provides software organizations a framework to improve their processes ([8], [9]). In SPICE, the analysis is performed in two dimensions: Process Dimension and Capability Dimension. Capability Dimension is divided into 6 levels and Level 4 (Predictable Process) has measurement (PA 4.1) and process control (PA 4.2) attributes. Measurement attribute requires a software firm to analyze trends in process performance and maintain process capability within defined limits across the organization. Similarly, Process Control attribute necessitates control of process performance to maintain control or implement improvement. The idea resembles that of Capability Maturity Model as the software firms are implicitly incited to use control charts.

The importance of quantitative methodologies is mainly because good decision making depends on how much you know about the process and the product. As defined in Metrics Guidebook of NASA [27], knowing what an organization does and how it operates is a fundamental requirement for any attempt to plan, manage, or improve. On the other hand, Tracey O'Rourke, CEO of Allen-Bradley, says

“without the right information, you’re just another person with an opinion.” Thus information is essential for understanding of processes and this leads to better management as soon as the information is valid.

In order to understand the need for measurement in project management, we can look at Project Management Body of Knowledge [30]. It defines project management as: “the application of knowledge, skills, tools and techniques to project activities in order to meet project requirements”. It investigates project management in nine knowledge areas and five process groups. PMBOK describes inputs, outputs, and relevant tools and techniques for the major processes in the knowledge areas in order to aid as a guideline for project managers. If we analyze Project Quality Management knowledge area, we see that control charts, pareto diagrams, statistical sampling and trend analysis are referred as techniques for Quality Control process. As PMBOK is a guideline, it does not have any obligatory nature for the software firms. However, it provides useful practices for good project management and SPC techniques are also recommended within this guide.

We see that CMM (Capability Maturity Model), its new version CMMI (Capability Maturity Model Integration), and SPICE implicitly direct software companies to implement Statistical Process Control as a crucial step for achieving higher process maturities. They suggest control charts for both project level process control and organizational level process improvement purposes. The traces of SPC are also found in software process guidelines such as PMBOK and NASA measurement guidelines. Actually, it is possible to utilize SPC methodologies to comply with some other software standards such as ISO 9000-3 [29], which requires determination of applicable statistical methods to monitor, measure and analyze the processes. This trend shows that software community has high expectations from statistical applications of SPC techniques. As seen, the results of practical SPC applications in the industry will be designative for its place in the future.

3.2. SPC in Software Development

As Card [2] 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. Fenton and Pfleeger [26] 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.

Humphrey [20] can be regarded as a reflection of quality management on software engineering 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 a 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 regarded in software industry, additional studies are being performed by the researchers. Lantzy [4] 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

Card [2] discusses the utilization of SPC for software by also considering some of the objections and mentioning about 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 [3] 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 [10] 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 software domain.

A similar work is performed by Florac and Carleton [12]. They 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 principles [24]. 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.

Radice [17] 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.

Weller [19] 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.

Jakolte and Saxena [16] 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 type1 and type2 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.

3.3. Contribution of Literature to Our Study

After going through the literature we realize that the current trends in software industry designate a movement towards utilization of SPC techniques. In CMM and even more in CMMI, SPC is depicted as a challenge for high maturity organizations. They are also the most detailed resources in terms of providing a guideline for SPC implementation in a software organization.

Apart from the software standards, the studies of researchers in this area also constitute an important part of the literature. One important common point among the researchers is that they all accept the difficulties of using SPC techniques in software domain. And a high majority of them regard SPC as a beneficial tool for controlling processes. While supporting their ideas, they usually refer to de facto experiences in manufacturing industries.

In some of the resources, we come up against examples of SPC for software metric data. Most of these examples are limited to defect density (defects / SLOC) and inspection effectiveness (defects / inspection hour) measurements. However, these studies contain very little information about implementation details. The characteristics that metrics should possess for proper SPC utilization, the categorization of data, the normalization of metric data to make it comparable among projects, and implementation procedures are not described. For instance, the control charts depict defect density data; but the definition of a defect, the type and priority of defects, the definition of SLOC, the components within SLOC counts (reused parts, blank lines, comment lines etc.), and many other small but very important details are not described. Because of these deficiencies, the existing studies are far from being capable of providing sufficient guidelines for

applying SPC techniques to software processes. Nevertheless, one can establish a good understanding of SPC before starting an improvement program in a software organization by reviewing the available literature. In this regard, our survey revealed that:

- SPC is not applicable to all software processes.
- SPC should only be applied to critical processes in a software organization.
- Not all SPC techniques are applicable to software processes.
- The processes should be well-defined and stable so that we can apply SPC techniques successfully.
- SPC techniques are required for achieving CMM Level 4.
- Control chart is the most sophisticated and useful SPC technique.

With these findings, we performed our case study in order to bring new insights on this issue.

3.4. Measurement Background

In order to establish a firm knowledge on the measures, we carried out a theoretical research at the beginning of our case study. We decided to work on rework effort, defect density, productivity and inspection performance metrics after long negotiations with the company (see section 4.1). We outlined the general properties for each measure including its definition, importance in software development, collection principles, various implications of metric information, difficulties in the measurement processes, data analysis guidelines, and possible uses of control charts relevant to the metric. In the following sections we first establish a firm understanding of defect and size measures, which form the basis for the issued metrics. Then we describe our findings that directed our way of analysis throughout the case.

3.4.1 Some Preliminary Issues about Defect

The number of defects in a work product is an important measure as it gives us an intuition about how much the customer will be satisfied (from post-release defects), how much rework will be performed, how efficient our inspection processes work, which processes need improvement, which components of the system are error prone etc. Therefore, defect counts provide evidence not only on the quality of the product, but also on the quality of the related processes.

Before starting to count defects in software products, however, it is important to establish necessary background in order to provide consistency among measurements. First of all, there is not an exact agreement on the meaning of defect in software industry. The terms defect, fault, failure and error are usually used interchangeably. In order to avoid such misconceptions and to establish a universal understanding, IEEE Std 982.2-1988 software standard [30] makes the following definitions:

Defect: a product anomaly. (1) omissions and imperfections found during early life cycle phases, and (2) faults contained in software sufficiently mature for test or operation.

Error: Human action that results in software containing a fault.

Failure: (1) the termination of the ability of a functional unit to perform its required function. (2) an event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered.

Fault: (1) an accidental condition that causes a functional unit to fail to perform its required function. (2) a manifestation of an error in software. A fault, if encountered, may cause failure.

From these definitions, we understand that defect is named as a fault when it is found in software, and a failure occurs when the software malfunctions as a result of a fault. Thus, not every fault results in a failure; and a number of faults together may cause a system/component to fail operation. Actually, it is common to come up against different definitions of these terms in different environments. However,

for ease of understanding, it is appropriate to refer to any software or documentation anomaly as a defect in our analysis.

If we consider an automobile, we see that it may encounter many problems with different criticalities, impacts, repair times and costs. If the breaks do not work properly, the result may be very serious, and even fatal; thus, the car should be repaired as soon as possible. However, a driver may use his car forever although the horn is not functioning. Therefore, a mechanic makes different treatments according to the characteristics of the problems. Software is similar to an automobile in that, each defect has distinctive properties. Not all defects have the same impact, urgency, origination point and cause. For this reason, it is not a good practice to compare software products only by looking at the number of defects without additional information.

In some cases, an anomaly about a functional software requirement might be regarded as a defect, although it may constitute more than one defect with different causes, priorities or sources. For instance, a defect detected by the customer during a qualification test may be recorded as "a new customer cannot be added by using addition function". From the customer's perspective, the defect is the inability to save a new customer record. When the problem is investigated, it may be realized that one of the fields (i.e. customer name) has a database connection problem, another field (i.e. customer NO) has a logic error while the confirmation button has a totally distinct problem. If these different defects are not clearly identified, the number of defects will give deviated results.

Another difficulty arises because of the necessity to distinguish between an enhancement and an error. Sometimes change procedures may not be appropriate to distinguish between an enhancement and a defect since both represent a change. Thus, a detailed conceptual analysis will be needed to decide whether a change is a defect or not, and most probably such a practice will not be feasible. Even with relevant change procedures, it will still be difficult to determine the reason of a change because of marketing issues. From the customer's point of view, a functionality may be regarded as an inherent property, for which there is no need to write an explicit requirement. So when that functionality is not present, he may

be reluctant to accept the product. The developer would not think this way, and may argue against the customer's intention. Most probably, the change will be performed by the software company considering the market conditions. Such cases are very frequently seen in software industry. And the nature of software development causes difficulties in determining whether such a change is an enhancement or a defect.

Benchmark

We see that, analysis and interpretation of defect requires some preliminary work and a precise understanding of some issues. First of all, a global perception of defect should be retained in order to avoid possible misconceptions regarding the defect definition. Secondly, we should categorize defects and products with respect to specific characteristics up to a certain level for meaningful comparison among metric data. Neither all defects nor all the products are the same. The defects can be grouped by type, priority and impact, whereas the products can be classified within certain product groups such as requirements documents, design documents, code, and test documents. It is still possible to divide groups into subgroups as soon as the data is available and the detailed analysis is feasible. Each software organization should individually determine the level of detail and the categorization of the groups after working on the historical data, determining specific needs and analyzing related processes. IEEE gives a practical framework for the classification of defect data ([31], [32]).

Moreover, the change procedures should be defined well enough to distinguish between an enhancement and a defect. It can be made either by using different processes and forms to implement an enhancement and a defect, or by depicting the type of change explicitly.

Having an understanding on **defect origins** beside the defective products would also be very beneficial for making a proper interpretation of defects. This information allows us to identify the cause of a defect and take corrective actions to improve the problematic process. A future defect may be a result of any previous process within the lifecycle. Thus, only the number of defects for a product may be misleading in terms of evaluating the quality of the related

process. By recording the origin of each defect, we may extend our analysis to investigate the defect injection potentials of different processes.

3.4.2 Some Preliminary Issues about Size

Size is one of the major and most frequently used metrics in software development. As it depicts the amount of production as a quantitative measure, it enables us to understand different dimensions of software processes and products. We can divide software process artifacts as documents and code. “Source lines of code” is one of the major measures for counting the size of code. However, some complexities make it difficult to measure software size in terms of SLOC [33]. First of all, there are many different definitions of SLOC in different organizations. Source code is composed of different sections like executables, data declarations and comment lines, and deciding which parts to include in counting is a hard issue for the software specialists. However, it is also important to provide consistency among different measurements so that they may be comparable to each other.

Another difficulty arises from the existence of different programming languages. The source lines of code required to perform a function changes from one language to the other. In order to avoid this problem, there are conversion tables that convert the number of lines in one language to the other. However, considering the defect density metric, it is difficult to decide whether to use converted lines or not. If a coding defect is due to a logical error, it would be reasonable to convert all source lines to assembly language. However, if the defect is due to a syntax error, the total lines of code would be more meaningful regardless of the language used.

Today many software tools provide enhanced capabilities for code generation, user interface development and compilation. Without writing a single line of code, it is made possible to create a user interface by dragging and dropping components. Integrated development allows software engineers to share their code with others and productivity is greatly improved. Although these additional features enhance software development, it makes that much difficult to find a concrete measure for quantifying software size. Comparing automatically

generated code with human generated code does not make sense as faults are always caused by human intervention. Shared code may also cause confusion as we may not exactly define the extent to which it is reused.

Complexity is also a critical issue in SLOC evaluation. It is most probable that a complex code component will be produced in a longer time and will have more defects than a simple one. Therefore, it is usually not reasonable to treat two source code measurements if their complexities are not similar. Actually, an alternative to physical SLOC measure is logical source lines of code, which can be more representative for the complexity issue. However, logical SLOC measure causes further problems regarding defect counts since defect may be due to syntax errors. Jones [34] outlines the main weaknesses and strengths of physical and logical lines of code measures. Although he reveals that logical statements are more rational for productivity and product quality measurements, he regards both of these measures as misleading. He finally suggests function point metrics for productivity and product quality measurements.

Finally, the reused code components cause problems in interpretation of defect density data. Ideally, a totally reused code is believed to be error free as it is inspected before. Practically, any code component has potential for containing defects, but considerably less in reused source statements. The extent and quality of reuse is influential on the number of defects we encounter.

In size measurement, Function Point is also used as an alternative measure for SLOC as it avoids the mentioned complexities. Function Point is a size measurement technique developed by Albrecht, who related the size of the software product to the required functionality. Thus, it is independent of how we develop the software. However, it is not easy to derive function points unless the requirements are exactly known. Moreover, the adjustment procedure is somewhat subjective.

There are different document size measures that are practically used in software industry. One of the frequently used document size measures is the number of pages. Although the measure is easy to collect, there are different concerns that we should consider before forming a relationship between the number of defects

and the number of pages. First of all, the measure depends on various physical properties such as font size, font type, page style, and margins. It would not be meaningful to compare two documents if one page in document A corresponds to two pages in document B. Secondly, a document is composed of different components like words, figures, tables and graphs. The complexity, size and defect containment potential of these components differ among each other, and among different documents. For this reason, the number of pages may mislead us while interpreting the defect density outcomes. One other concern should be the reused and nonfunctional parts of the documents. Many of the project documents have sections that are similar in content. Therefore, it is possible to reuse those sections. Moreover, the empty pages, cover page and pages including table of contents, index, signatures etc. are not apt to have defects.

Another way of representing document size is the number of words. As it undermines the necessity to consider physical aspects of a document, it may be more appropriate for analysis. However, the concerns about reused sections and document components other than words are still existent.

Benchmark

We see that the measurement of size has many inherent difficulties. So we have to think on the mentioned considerations before starting a measurement process.

Firstly, the size measures should be determined very carefully for different products or product groups. In order to provide a reasonable basis for this decision, the existing products may be investigated and various product size measures may be tried. Finally, a specific size measure, which most appropriately represents the size as a direct proportion of the number of defects, may be determined for each product or product group. For instance, it can be lines of code for software; number of pages or number of words for document; function points for the whole project.

For code defects, definition of SLOC should be precise for each count. If line of code data is available for software size, non-comment, non-blank lines of code should be counted. In order to avoid any deviation in the results, it is possible to exclude reused code from the dataset. However, the inferences may deviate when

a defect is correlated with the excluded code component. Jones [34] delineates that the quality of reused components has an impact on software productivity in the range of % 650. In order to get rid of this variance, some percentage of the reused code should be added to total measurement data considering the amount and extent of reuse. Likewise, not all, but some percentage of tool-generated code should be counted, depending on the respective amount of effort it takes to generate that code. It is not straightforward to determine the exact values for these percentages. At this point, the most important asset of a software organization turns out to be its historical data. Different percentages may be tried and the values that produce the most meaningful outcomes can be selected.

Separate analyses might be performed for the measures in different programming languages. If there is not enough data to perform separate analysis, converted SLOC count should be used for logical and algorithmic code defects, whereas unconverted SLOC count should be used for syntax related defects. Obviously, such an analysis necessitates a detailed categorization of defect data.

In order to refine the analysis, it is also possible to include the complexity dimension by normalizing the SLOC count by multiplying with a complexity adjustment factor (such as cyclometic complexity). However, we should be aware that including so much detail may cause frustration as we may get lost in the complexity of data. We can think of countless more factors that may be influential on the size of a code component. The important point is to establish a balance between the amount of detail and the cost of this detail in terms of data collection effort, understandability and analysis difficulty. We should have enough detail so that the outcomes produce meaningful results, but we have to avoid redundant detail in order to prove a feasible analysis.

For measuring document size, it is better to use different size measures for different documents. For requirements specification document, the number of requirements is a good measure to represent the functional size as we will relate it to the effort expended to produce that document. Yet, it is difficult to define a similar specific measure for plan and design documents. Thus, the number of pages or words may work better. In either case, however, it is possible to refine

the data by using further normalization techniques. For instance, different weights can be used for functional and non-functional requirements in an SRS document. Likewise, figures and graphs can be added to the calculation with relevant coefficients. The level of detail should be determined in view of the criticality of the process/product.

In order to have a different view on size, function point can also be used that gives the functionality of the product. Function Point is a relatively robust measure as it is independent of specific code and document size attributes (such as programming language, software tool, reuse, complexity (for code); physical layout, reuse (for document). If we have sufficient evidence to propose a direct proportion between the effort spent in different project phases or the size of a work product and the FP count, we may utilize this information for our analyses. However, the pure function point count should be adjusted in order to account for technical complexity and certain quality requirements of the application. This is performed by multiplying the FP count by adjustment weights that are determined by instantaneous factors. Moreover, the function point estimation should be performed by experienced individuals to minimize the variation due to subjective decisions. Nevertheless, the limitations regarded with FP based analysis should be considered specifically for each metric (defect density and productivity).

3.4.3 Rework Effort

Before measuring rework effort, it is essential to have a clear understanding of what is rework and what it is not. NASA software measurement guidebook [27] defines rework as “the total hours spent that was caused by unplanned changes or errors”. According to the definition, unplanned changes are referred as rework. However, some of the changes may be additional customer requests that can be regarded as enhancements. Considering these enhancements as rework or not depends on our aim of measuring rework effort. If we want to take care of the amount of deviation from the plans, enhancements can be included in rework effort. However, if our aim is to measure the work performed as an extension of previous work due to errors, enhancements should not be included in rework calculation. In this study, our focus is more on the effort expended to fix defective

and missing parts of previous works. Thus, we will use a modified definition by excluding enhancements.

The amount of rework is a good indicator for the quality of processes as it shows the magnitude of effort we spend due to previous errors. Rework increases project costs without adding any value to the product. Thus, it is possible to produce a product without any rework by doing the processes right the first time [5], which is described by "Cost of Quality". Houston [35] categorizes the cost of software quality into two groups: costs due to lack of quality and costs of achieving quality. Rework constitutes the major part of the first group. Krasner [36] demonstrates the relationship of CMM maturity levels to rework rates and expected quality levels according the studies on Lockheed's projects (See Figure 3.1). We understand from the data that the rework percentage decreases and the product quality increases as the processes become more mature. This also supports the idea of CoSQ by showing evidence for the benefit of process improvement programs.

Process Maturity (characteristic)	Rework (percent of total development effort)	Product Quality (defect density)
Immature	>= .50	double digit
Project Controlled	.25-.50	single digit
Defined Organizational Process	.15-.25	.X
Management by Fact	.05-.15	.0X
Continuous Learning and Improvement	<= .05	<.00X

Figure 3.1 CMM Maturity vs. Rework and Product Quality

Actually, we can have an idea about the quality of a process by measuring the number of defects in the related work product. However, defect measure does not give evidence for the magnitude of the effort needed to fix those defects. Even though we can categorize defects within certain impact groups, this necessitates a detailed measurement background and the data will be in ordinal scale, which will be less descriptive for analysis. Rework directly demonstrates the impact of defects beside their quantity as its focus is on effort value. As a result, rework effort and defect counts can be regarded as supplementary measures for analyzing the quality of software processes and products.

Although the idea is simple, it is natural to encounter some problems when we want to implement rework analysis in practice. First of all, the measurement of rework effort is quite difficult. Defect correction times are usually recorded on trouble reports as part of problem resolution procedures. However, people are usually inclined to show their rework times as low as possible since rework is believed to be an indication of poor performance in earlier phases. For this reason, the collected data may not represent the actual amount of rework occurred. It is also possible to gather rework effort from timesheet data if the timesheet activities are detailed enough. Nevertheless, the same problem still exists and data maintenance will also be more arduous.

Secondly, the cause of rework may not be traceable directly from the data. A requirements error may lead to design and code updates, which are accordingly treated as design and code rework. Therefore, high rework amounts related to a specific process may not be indicative for a deficiency in that process.

Finally, rework effort does not mean so much by itself. The number of defects is mostly associated with the size of the related activity. We expect to have more defects during a comprehensive process, and few defects in a small process. As defects are sources of rework, rework effort should also be related to the size of the corresponding process. Thus, we may utilize a normalized measure in order to provide a baseline to make a comparison among rework efforts of different activities. And rework percentage acts as a meaningful metric for this issue, where:

Rework Percentage = (Rework Effort) / (Total Effort)

for the related activity.

Benchmark

Rework percentage gives us an understanding about the relative amount of rework with respect to the total effort. Thus, we can make an analysis for any group of tasks representing a specific domain. For instance, the performance in a software domain (i.e. quality assurance, configuration management etc.) comprising particular tasks, the quality of activities in a certain project phase or the processes during the preparation of a work product can all be examined by looking at rework percentage values. However, proactive actions should be taken to avoid the above mentioned problems.

First of all, a company-wide perception should be provided for the importance of valid rework effort. People should be aware that the rework data is used to improve processes, but not to evaluate or even blame them. In this way, the tendency to hide rework efforts can be avoided.

The meaning of rework should also be clearly defined. The means to measure rework effort and total effort should be established. The procedures should be documented so that enhancement efforts are not regarded as part of rework effort.

During an SPC analysis, rework percentages might be compared among different projects, CSCIs and builds within specific time intervals. In this analysis, the focus will be on analyzing the rework percentage related to a specific project phase among different CSCIs/projects. Thus, we may interpret rework percentage as the amount of rework per unit effort, where the effort is different among different projects and CSCIs. Considering the definition of our metric and the individuality of the measurements, XmR chart will be the right option for SPC analysis.

A sample Individuals chart is given in Figure 3.2. In the graph, the points above the upper control limit indicate problems in the design phase of Project 3. This may be because of allowing too little time for design so that many updates and additions were necessary during later phases; or the design was carelessly

performed and this caused corrections when the deficiencies were detected in later phases.

We see that Figure 3.2 is not very effective to provide on-time process tracking since the graph can only be drawn at the end of the related project phase. However, the root causes can be analyzed for the special instances and the results can be utilized for process improvement studies. Moreover, the analysis can also be expanded to whole project level for this aim.

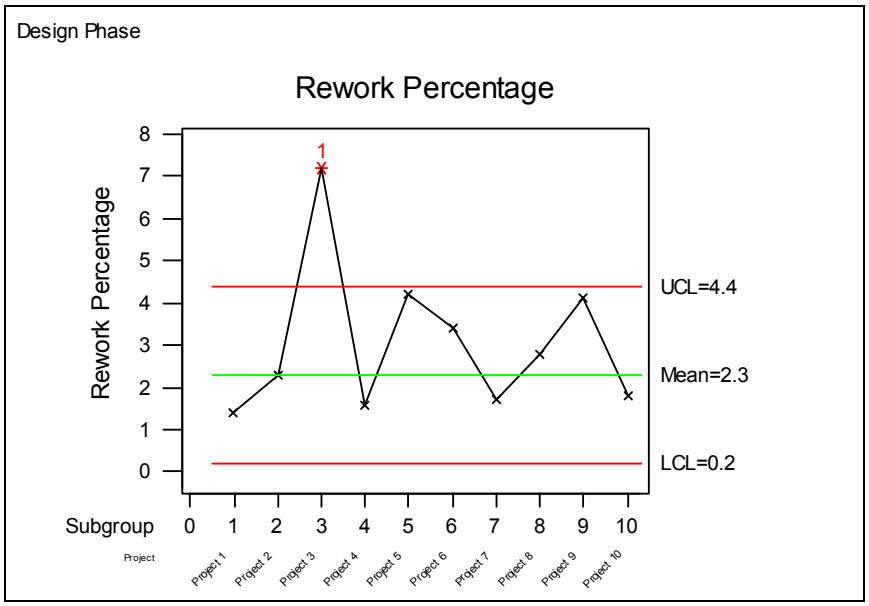

Figure 3.2 Sample Individuals Chart for Rework Percentage (Phase based)

An alternative approach to provide continuous control during process execution can be a periodical rework analysis. The rework percentage may be calculated on a periodic (i.e. monthly) basis independent of the project phase. According to the quantity of data on hand, the analysis may be carried out separately for different CSCIs and/or builds. Considering the dynamics of a software project, we expect to have higher rework towards the end of project phases due to the errors found

during inspections. Thus, this analysis should be based on the assumption that the relative rework amount is constant through a high-enough period of time. For this analysis, Individuals chart can be utilized (See Figure 3.3).

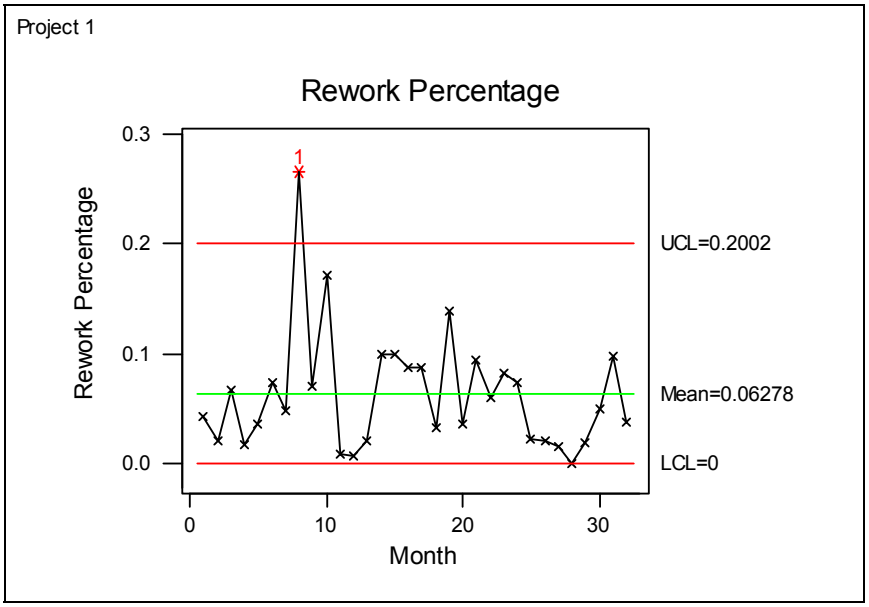

Figure 3.3 Sample Individuals Chart for Rework Percentage (Monthly)

In this chart, the control limits should be calculated using the data of all the projects in order to view the status of processes in organizational level and to provide tracking since the first phases of each project. The values exceeding the upper limit may be indicative for deficiencies in project planning, bad performance of processes, high inspection performance or a long period of testing and inspection processes. The points under the lower limit might be due to high inspection performance, superior process performance, or a very long project phase (We know that most of the rework is done towards the end of project phases after inspections. If a project phase lasts too long and the inspections have not been performed yet within the measured period, the rework will accordingly be

low). On the other hand, an increasing trend may be due to problems in **earlier** project phases or **previous** inspection processes. In this case, more emphasis should be given to ongoing inspections as an aid to avoid future deficiencies. A decreasing trend may be the reason of bad performance in current inspections, or might be indicative of high product quality. Obviously, rework percentage and defect density charts should be analyzed concurrently so that more reliable interpretations will be possible.

It is also possible to analyze rework from a different perspective by finding the amount of rework **related to** a certain process or project phase. For example, all the rework effort in different phases related to requirements analysis phase can be compared to total requirements analysis effort (including reworks in later phases). The formulation then becomes:

Rework Percentage = (Total rework effort related to phase X) / (Total effort for phase X)

In this case, we should work on each project phase separately. A sample Individuals chart showing the rework percentage amounts for 15 projects/CSCIs is given in Figure 3.4.

The chart depicts the rework percentage amounts that are related to requirements analysis phase based on measurements at the end of design phase. The analysis for requirements analysis rework should be repeated based on the end of coding, testing and maintenance phases. The same analysis should also be performed for the project phases other than requirements analysis.

Obviously, such an analysis necessitates the determination of defects' causes. However, the results will give an additional understanding of the quality of software processes. Not as a substitute, but as a complementary analysis to rework percentage, this study can be very beneficial for project planning and improvement purposes.

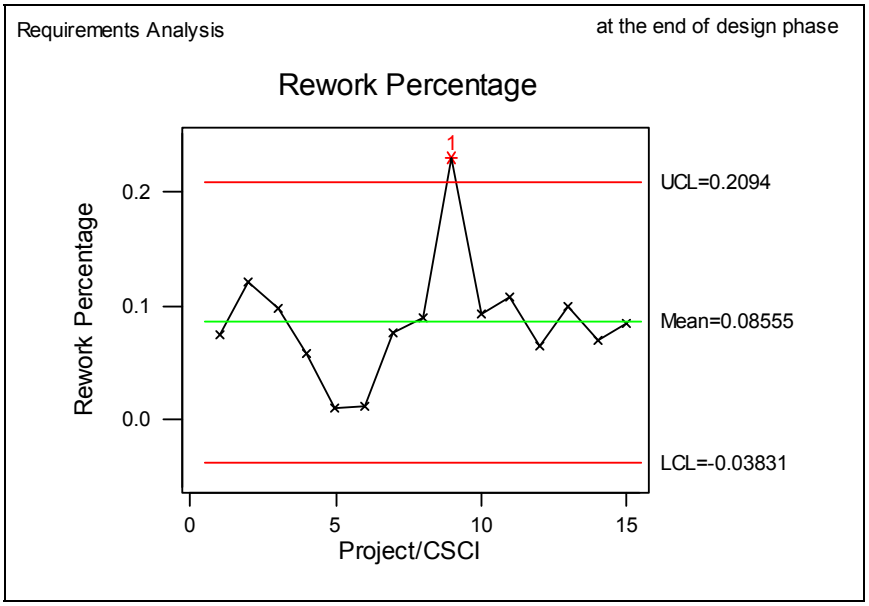

Figure 3.4 Sample Individuals Chart for Rework Percentage

3.4.4 Defect Density

In the previous sections, we mentioned about the difficulties of measuring defect data in a software company (section 3.4.1). But once the relevant process baseline is established, the software firm can have an opportunity to analyze detected software defects to visualize process and product quality. However, the number of defects found in a document or code component is not very meaningful by itself. There should be a basis for comparison between different measurements so that a specific measure can be meaningfully interpreted. Table 3.1 shows this situation by giving the number of defects in three components. At the first sight, component 3 seems to be the most defective, while component 1 the least defective. However, without the knowledge of component sizes, we cannot say whether it is abnormal to have 1123 defects in component 3. At this point, defect density measure provides another representation of defect data by taking into account the size dimension.

Table 3.1 Defect Count

	Component 1	Component 2	Component 3
# of defects	23	123	1123

Defect density is defined as:

$$\text{defect density} = \frac{\#of.defects}{product.size}$$

The analysis and interpretation of the metric data lies on the assumption that, on average, we have a certain expectancy of defect count for unit size of a software artifact. As we desire to find most of the existing defects in a successful inspection, this measure is also an indicator for the effectiveness of the related inspection.

The addition of product size beside defect count increases the complexity of our data collection and analysis processes. There are various software products and size measure of a product will not be appropriate for the other. For instance, it can be lines of code for software; number of pages or number of words for a document; function points for the whole project. Although defect density metric neutralizes the effect of size on defect count, there are still other parameters that should be considered in order to provide a firm basis for data interpretation. The preliminary issues about measuring size for software artifacts are discussed before (Section 3.4.2). The mentioned difficulties are also existent for defect density analysis. The reason to mention about these influences is that, SPC requires rational sampling of data. If the data samples have different distributions, the variance will be high and the sensitivity of the control chart will be greatly reduced.

Benchmark

We see that, analysis and interpretation of defect density metric requires some preliminary work and a precise understanding of some issues. Once the mentioned

preliminary issues are resolved (as defined in sections 3.4.1 and 3.4.2), it will be possible to perform reliable SPC analysis for defect density.

In most of the resources, u chart is suggested for tracking defect density data ([10], [17], [19] [26],). Actually, u chart is specifically designed for depicting the number of defects per inspection unit with variable sample size. As different software products have different sizes, u-chart seems to be an appropriate analysis technique for defect density data. However, it depends on the assumption that the defect data follows a Poisson distribution. Therefore, it is very important to check for the validity of this assumption by analyzing metric data.

We know that the width of control limits for a u-chart is inversely proportional to the square root of sample size (see section 2.1.3). In defect density analysis, the sample size is defined as SLOC for code and number of words/pages for document. Based on the idea of u-charts, having 10 defects in a sample of size 20 is not equivalent to having 100 defects in a sample of size 200. This is reasonable when we think of a sample as the number of items for which defect is counted (i.e. a sample of n metal sheets is taken and total number of defects is counted). In this case, if the sample size n is small, the variance will be high because an individual shift will be more effective in the sum. Nevertheless, the validity of this reasoning is questionable in software defects. When we take SLOC as sample size, we inherently make an assumption that each line of code is inspected, which is usually not true. Moreover, the idea that control limits extend by lower sample size for a u-chart seems invalid when we consider thousands of code lines. A similar reasoning can also be deduced for document type software products. Actually, we expect to have the same limits for two equal defect density measures which have different sizes (i.e. 100 defects in a size of 10000; 200 defects in a size of 20000).

Another issue for the u-chart analysis is the assumption that the data is Poisson distributed. In most situations, we do not have sufficient evidence for the validity of this assumption. Summing up all these constraints together with the variability of software data, we prefer to use XmR charts for defect density analysis. However, it may still be possible to benefit from a u-chart if SLOC is counted for

software units that have similar sizes. In either case, the observations that exceed the upper limit may indicate a highly defective component, or a very successful inspection process. Likewise, a low defect density measure may indicate a high quality product with very few defects, or a poor inspection process. By charting the data in time order, it is also possible to see the trend of defect density through a period. Obviously, there is always a risk that the deviations are due to measurement or calculation errors.

Apart from defect density, defect data can be used for further analysis in order to provide control for some other processes. For instance, *man-hour/defect* measure gives us an understanding of the efficiency of problem resolution process, where man-hour corresponds to the time spent for fixing defects. Likewise, *SLOC/hour* [27] measure may be utilized to understand inspection process effectiveness, where man-hour signifies the time spent for code review. However, similar pre-analysis studies should be performed to determine measurement parameters as described in sections 3.4.1 and 3.4.2.

3.4.5 Productivity

In general terms, productivity means the amount of output produced per input expended. The amount of output can be the number of painted Coke bottles, weight of refined petroleum, or length of pressed metal sheet. Input measures can also take different forms such as weight of paint used, amount of catalyst added for refinement, or amount of electrical energy necessary to activate pressing machines.

In software development, the main outputs are the work products, namely code and documents. And size measure is used in order to quantify the amount of work product produced. On the other hand, the major input for the production of work products is human resource. And effort is used to quantify the amount of human resource utilized. After all, the productivity measure turns out to be the size of the work product produced per unit effort.

Productivity is an important and critical measure as it gives direct implications about how efficient we perform our processes. We use our productivity data to

make reliable plans, to detect any deficiencies in our processes and to visualize the impacts of improvement actions. It is also a key measure for top management since higher productivity brings lower costs, increases the opportunity for profits, and improves competition force in the market.

Although the definition seems simple and straightforward, it is actually difficult to set up a firm basis for measuring software productivity. We typically expect the amount of output to increase as we increase the amount of input. In manufacturing applications, we can propose a direct proportion between inputs and outputs. Thus, regardless of the output size, productivity gives a clear idea about how efficient we perform our processes. Unfortunately, this is not the case in software development. A code component that is built in two days by an engineer may not be built in one day by two engineers. Thus, there is a diminishing rate of marginal return on inputs. For this reason, we may expect different productivity levels for different output-input combinations.

We know that productivity is a means of capturing the relationship between the effort expended and the value of the corresponding outcome. Thus, our size measure should represent the logical value of the artifact. However, the nature of software work products makes it difficult to obtain a suitable size measure. SLOC is the main measure we use for software size but it includes inherent parameters as defined in section 3.4.2. Jones [34] states that using lines of code for productivity studies involving multiple languages and full life cycle activities should be viewed as professional malpractice. Similar problems also exist for document type work products. In manufacturing, the number of artifacts is a good indicator for the value of the output, since different samples include the same products. In software development, however, each document is unique. Therefore, it is questionable whether the physical measures like “number of pages” or “number of words” correctly represent the actual value of the work product. A more detailed analysis of document size is given in section 3.4.2.

Another difficulty arises while measuring the effort amount expended for the related work product. For instance, the preparation of an SRS document necessitates a number of activities including requirements analysis, review,

training and final documentation. People from different functions and with different responsibilities are involved in these activities. Thus, collecting and distributing the relevant data to the related work products correctly is a difficult task.

Benchmark

Productivity is one of the major indicators for the performance of an organization and managers always pay special attention to their productivity levels. Software industry is not an exception and the complexity of software development further increases the importance of productivity. For this reason, productivity measurement has an exceptional consideration in software community despite the mentioned difficulties.

In most of the productivity measurements, software code is regarded as the only product for the projects. In this case, the analysis can be performed at the end of the project and continuous process tracking becomes impossible. Nevertheless, many document artifacts are also produced throughout the life cycle of a project. And we can propose that the effort spent in a project phase is directly proportional to the size of the related document produced in that phase. After making this assumption, we may track the productivity for each project phase and take relevant actions during project execution.

In order to gather beneficial outcomes from productivity data, the data collection and analysis procedures should be carefully established. Productivity data should not be used to visualize personal performance of the engineers. Sometimes, managers feel the need to evaluate the performance of their personnel on a concrete basis, and have an inclination to use productivity data for this aim. However, this may be a dangerous attempt as any measurement is under human control. If a person realizes that her performance is evaluated by different measures, she can pretend to work productive so that productivity values turn out to be high. For instance, longer algorithms may be used to inflate source lines of code data, although the functionality remains the same. This not only deceives actual process performance, but also causes a mood of frustration among the personnel.

Once the scope of productivity measurement is drawn, the procedures for the detailed analysis can be determined. Now we will go over the mentioned difficulties one by one.

Firstly, size definition and measurement procedures should be established considering the mentioned difficulties (section 3.4.2). Once this is done, different size measures (FP and code/document size) can be used concurrently to have a deeper understanding on the processes.

An effective effort counting technique should also be utilized to gather relevant effort data in required detail. Summing up all the efforts related to a work product necessitates a well-structured process baseline as well as a detailed and meticulous data collection mechanism.

We know that, the total effort to finish a piece of work differs according to the number of personnel used. Thus, the productivity is dependent on process execution and we observe variation in measurements. Actually, it is quite frequent to see unbalanced work assignments in software companies because of contract constraints and difficulties in project planning. Thus, the range of our control limits may become too large causing the control charts to become insensitive to changes. In order to use the control charts more efficiently, we can suppose that this variation is natural, and we can normalize our productivity data in a way to smooth out the effect of the number of personnel in the processes. This can be done by multiplying our productivity measures by a normalization factor such as:

$\alpha / \psi(n)$,

where α is a constant, n is the number of personnel and ψ is a function of n. In order to find the effect of n (independent variable) on the productivity (dependent variable) we can run a regression by using historical data. In this analysis, it is important to decide whether we want to catch the variation due to the number of personnel as an assignable cause.

Even though the mentioned issues are taken into account, it is still difficult to obtain reasonable results from an SPC analysis on productivity measures. Jones [34] lists the factors that are positively and negatively affecting productivity data,

where the total impact ranges up to % 1867. The SPC outcomes for such a variable process should be interpreted with great care, and should not be taken as scientifically proven facts. The analysis, however, may be utilized in providing an additional eye on the software processes.

Sample individuals control chart for SRS productivity data is shown in Figure 3.5. 10 SRS documents are from this year's projects, but the average values and the limits are calculated by using all historical data.

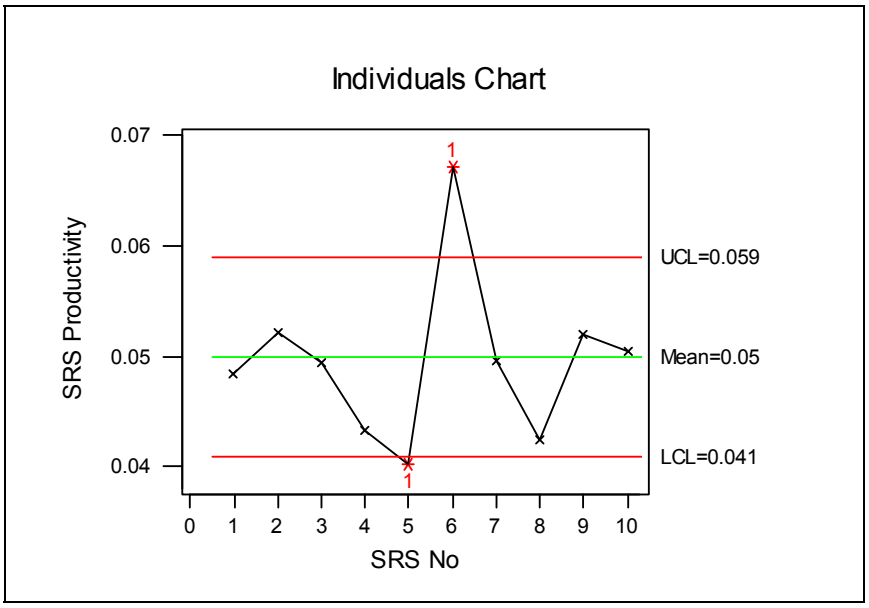

Figure 3.5 Sample Individuals Chart for SRS Productivity

In the analysis, the observations that exceed the upper limit may indicate a very effective study or a hasty requirements analysis process. Likewise, a low productivity measure may indicate an inefficient requirements analysis or a very meticulous, complex or detailed study. On the other hand, if the SRS documents are timely ordered in the graphs, an increasing or decreasing trend can also be recognized. In this way, we may be able to visualize the effect of any

improvement studies. After detecting the outliers, further analysis will be necessary to discover the causes of deviations and take corrective actions if necessary.

Even though the SPC analysis can be improved by the mentioned practices, the data may still be too variable to provide a successful control mechanism. The data on hand may not also be quantitatively and qualitatively sufficient to provide satisfactory outcomes. Because of the possibility of such deficiencies, we should be careful on applying and interpreting control charts.

3.4.6 Inspection Performance

Inspection is the formal review of a product in order to identify defects. It enables software specialists to visualize and correct their errors before product release. The practical experience shows that inspection is a powerful tool for improving product quality and for reducing rework amounts. It is also used to provide an agreement among appropriate parties, to verify product acceptance criteria, to complete a formal task and to provide data on the product and the inspection process [20]. Moreover, the cost of fixing a defect in a system test is about ten times more than the cost of fixing the same defect in requirements review [38] (see Figure 3.6). This makes it critical to detect and correct defects as early as possible.

The importance of inspection on product quality and project costs necessitates tracking the performance of inspection process in a software company. Different metrics such as average preparation rate, average effort per product size, average effort per fault detected, percentage of re-inspections, and average faults detected per product size are suggested and used in industry. Actually, it is possible to obtain various data for inspection process. However, the data should be carefully defined and collected so that the outcomes may be meaningfully interpreted.

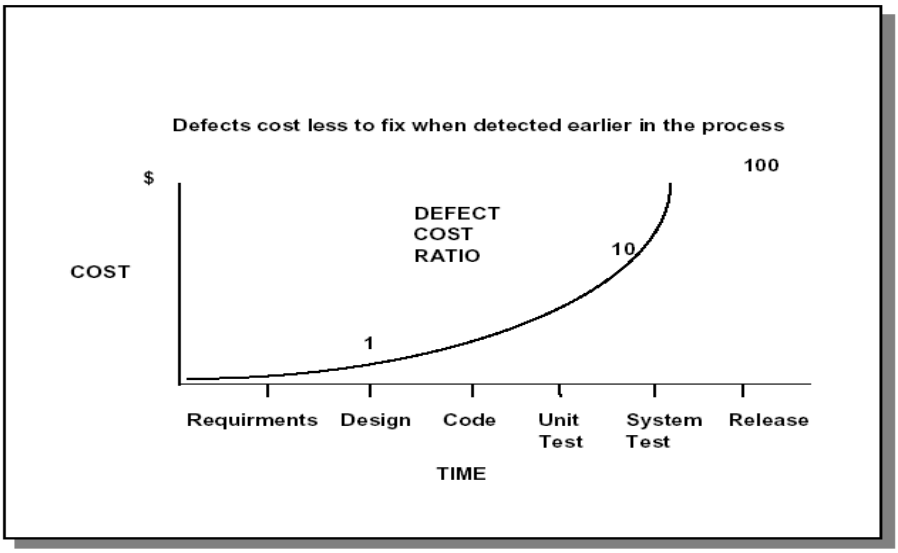

Figure 3.6 Defect Cost Ratio

One of the useful measures reflecting inspection process performance is the inspection effectiveness:

Inspection Effectiveness (1) = (# of defects found) / (inspection effort)

If we consider the number of defects found as output of inspection process and the effort as the input, the metric turns out to be inspection productivity. The interpretation of the data depends on the idea that the number of defects found during an inspection is directly proportional to the inspection effort. Thus, we expect to find more errors if our inspection lasts longer.

Our focus in this analysis is on review process, which is dependent on review procedures, the reviewed software product and the reviewers. Moreover, the reviewers will usually be from different project groups or components for independent verification purposes. Therefore, there is no need to perform separate analyses for different components (CSCIs) or builds. However, the review performance should be analyzed through a time period so that the trends can be visualized as the review process improves.

Before analyzing inspection effectiveness metric, we have to mention some of the related concerns. First of all, we have the same difficulties of measuring the number of defects as we mentioned in section 3.4.1. Additionally, it is probable that the same defects be found by more than one of the reviewers in an inspection. In such cases, the detected defects include duplicates and only one of them is recorded. So, although the inspection effectiveness of an individual may be high, overall effectiveness appears to be lower. For this reason, we expect higher inspection effectiveness when the number of reviewers is low.

Likewise, the measurement of inspection effort is apt to cause confusion and misinterpretation. The review process includes the assignment of reviewers, the active review of individuals, the documentation of review results and their exhibition to the authors in a meeting. Which actions to include in total review time is a difficult decision and should be determined by thinking on outcomes expected from this metric.

The characteristics of the inspections are also influential on the outcomes of the metric data. Naturally, the number of defects found during an inspection depends on the quantity of defects present in the product. Before the initial release, the product contains most of the defects that it will possess through the whole life-cycle (all except bad fixes). Thus, the initial review of a product just before the first release is apt to capture many defects. In the subsequent phases, the newer defects are injected during fixes and changes, which constitute a small portion of the product. Consequently, the number of defects in the product decreases as the product matures. Similarly, most of the inspection process definitions necessitate a final check for the correction of the defects that are detected in a previous review. And it is uncommon to find additional defects during these reviews. Joint reviews are also believed to be error free as the product is reviewed and its faults are corrected before submission to the customer.

One other important point that we should consider is the product under consideration. The inspection effort and the related number of defects may typically display discrepancies among different products. For instance, most of the plan documents contain duplicate information and once a plan document is

released, the others reuse reviewed parts of the released plan document. Therefore, we expect relatively low defect levels in plan documents considering their size. Likewise, the approach for inspecting a code component is quite different from reviewing an SRS document, which contains different sections such as product overview, functional requirements, flow diagrams and Use Case diagrams.

There are other measures that give further idea on the performance of inspection process. One of them is the inspection effort per product size. This measure provides another dimension with the assumption that it takes more time to inspect products having greater sizes. These two metrics together with defect density are complementary, and give a complete understanding on the product quality and the inspection process performance.

Benchmark

In order to maximize our perception of the metric data, we have to establish a common understanding on some preliminary issues.

First of all, measurement of defect should be well defined in the software organization (see section 3.4.1). As the metric relates the number of defects to the inspection effort, it is also important to calculate effort amount correctly. Since the detected defects are discussed and verified in a review meeting, the man-hours spent in a review should also be included in the total review effort amount. The collection of this data may necessitate a mechanism through meeting minutes or specifically designed forms.

While performing statistical analysis, it is important to provide various data relevant to the process for appropriate interpretation. Although the data may not be utilized directly, it may be applicable for data categorization. Considering that the number of people may affect the effectiveness of inspection process, it may be a good action to record the number of reviewers for each review. In this way, we may have an opportunity to visualize its effect in more concrete terms. We may even perform a regression analysis by relating the inspection effectiveness to the number of reviewers and normalize the metric to obtain more meaningful results.

As mentioned before, we expect to capture fewer defects in the later reviews. However, it is also likely to spend less effort during inspecting products after first release since the focus is mostly on changed sections. Therefore, we may start with the assumption that the proportion of defects to inspection effort stays at a certain level. By working on the metric data, we may develop normalization methods by including inspection sequence as an additional factor in the analysis. Moreover, it is possible to perform separate analyses with respect to the order of inspections.

As the inspection outcomes depend on the product, it is best to analyze metric data separately for different products or product groups. For instance, requirements documents (SRS and IRS), design documents (SDD, IDD), plans and code may constitute analysis units. Nevertheless, these groups should be formed specifically in each company considering the related standards and procedures. Moreover, the data may be analyzed to have an understanding on which products show similar distributions.

As soon as the mentioned considerations are handled, the data can be used for SPC analysis. For the individual inspection effectiveness measures, XmR charts can be utilized. A sample Individuals chart regarding the inspection of requirements documents is given in Figure 3.7.

In the figure, the points above the upper control limit show the instances in which the number of defects found per unit effort exceed the process performance limits. This may be because of high inspection effectiveness or low product quality. On the other hand, the points below the lower control limit may be indicating low inspection process performance, when there are many defects left undetected on the product, or high product quality when the product actually contains very few defects. In either case, the defect density measure may be used in parallel to reach the correct understanding while interpreting the results.

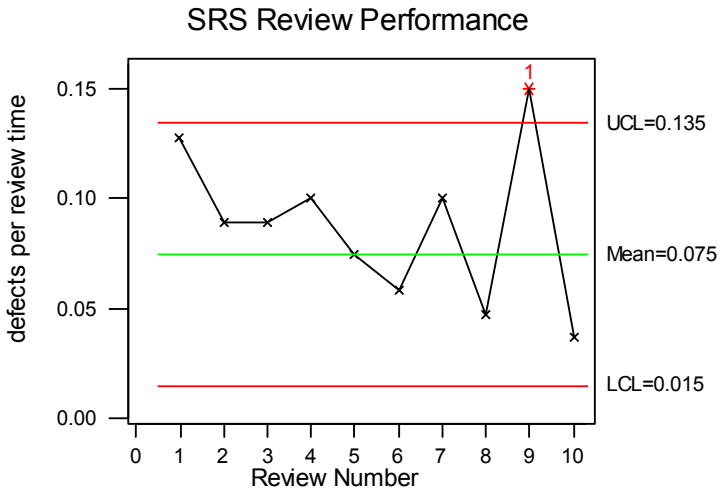

Figure 3.7 Sample Individuals Chart for Review Performance

Actually, it is also possible to use u-charts for a more sensitive analysis by regarding every minute spent on inspection as a size unit for finding a defect. However, the applicability of such a sensitive analysis should be investigated by using the existing metric data.

CHAPTER 4

CASE STUDY

4.1. Fundamentals of the Case

In the previous sections we introduced the context of process control; described how statistical techniques are referred in software process standards and software process guidelines; explained Statistical Process Control and demonstrated the means of applying control charts to software processes. We also gave a summary of research studies on the subject and our studies showed that there are very few studies demonstrating details of utilizing SPC techniques in software domain.

The need for such knowledge in the industry encouraged us to perform a study to show how SPC techniques can be utilized in software development life cycle processes. We started with the idea that it is possible to use control charts for software processes with specified metrics and we aimed to explore whether SPC can produce beneficial results for a software company. In the best possible conditions, we could find a software firm with well-defined processes, a high number of projects and a strong background on measurement. We would then define specific metrics and collect data in detail for a long period of time. By analyzing the data using control charts, we would evaluate how successful SPC is in detecting process nonconformities and in visualizing process behavior. We could also find out the effects of different metric parameters (i.e. the reuse percentage in SLOC count), and different control charts on the results by performing an experimental approach. At the end of the study, thus, we could understand for which processes SPC is meaningful; which metrics represent the

desired stable characteristics of the process; how the metrics should be normalized; and how the charts should be interpreted.

In our study, the primary constraint was time. As part of a master's thesis, we had about ten months to finish our work. Therefore, we preferred to base our study on the existing data of a software organization instead of waiting for defining metrics and collecting data. This limited our influence on metric data and prevented us from performing an experimental analysis. Considering its advantages on the analysis of real-time processes, we selected "case study" as our research strategy.

In order to set the boundaries of our case study, we initially determined the questions to be answered. As a result of this study, we expect to find out:

1. What are the difficulties of using the existing metric data in order to apply Statistical Process Control to a software company that has well-defined processes?
2. How can SPC be applied to the specific software metrics?

In order to give a direction to and narrow our analysis, we proposed that SPC can be applied successfully to some specified processes if

- the related process is stable,
- various factors within the process are defined and reflected as parameters in the metrics.

As we started with the idea that the processes should be stable for successful implementation, we selected a CMM level 3 software company to perform our case study. The company has 56 technical personnel and is experiencing a period of improvement since its foundation. As a result of this continuous improvement period, it is certified by CMM in August 2002 and further studies are currently in process to achieve CMM Level 4. Considering these specific characteristics, we defined our scope of case as a representative study for small-to-medium scaled, improving organizations, which we called "Emergent" organizations. Nevertheless, most of the mentioned concepts can also be generalized for any software organization.

The company has been involved in industry since 1998 and measurement is given special importance for process tracking and improvement. Currently more than 20 metrics are being tracked including effort, rework effort, defects, requirement stability and SLOC. Many other quantitative data are also collected via forms and reports. Although some of the data were being collected before achieving CMM Level 3, process definitions existed before the measurement processes were performed. Therefore we can regard the software processes stable.

Before starting to work, we made an agreement with the company to outline the boundaries of our study. A statement of work is prepared and signed by both parties (Appendix A). Moreover, we prepared a proposal to document the general objectives of our study. It is decided that:

- the name of the company will not be mentioned in any part of the study considering the confidentiality of the utilized data
- although the actual data will be used, the scale of the data will be modified without changing the trends for confidentiality purposes

Throughout our research, we performed a collaborative work with assigned company personnel. We provided the discipline as if we were taking part in a commercial project. We performed regular meetings in order to decide on different issues, to lead our study and to discuss our findings (see Appendix B to see meeting times and subjects).

We started our study in two dimensions. On the one hand, we prepared a list of all metrics tracked in the company and worked on each measure considering its meaningfulness and feasibility in terms of performing SPC analysis. We used such a bottom-up approach to select the metrics among the ones that are already being collected in the company.

On the other hand, we tried to figure out the most critical issues that need close tracking in the company. Consequently, our primary focus has been on the factors that influence product quality, project cost and schedule. We regarded the following items as critical for our statistical process control purposes:

Rework: As rework is a non-value-added effort, we can not only reduce the project costs but also obtain shorter schedules by controlling and minimizing rework effort.

Inspection: Rework is a natural result of nonconformities in project artifacts. Moreover, defects in a product are indicators for low quality. Since the inspection process is the major control point for the defects, we can reduce rework effort and improve product quality by improving the inspection process.

Requirements Analysis: We know that requirement problems have the most significant negative impact on project cost and schedule. Thus, it is critical to track requirements analysis processes in order to obtain a healthy project life cycle.

Afterwards, we performed a mapping between the collected metrics and the selected processes. As a result of this mapping, we constructed a list of measures as initial candidates for our analysis (see Table 4.1). Although these measures were not directly collected, we had the relevant data to derive them.

After an initial evaluation, we decided to eliminate some of the candidate measures which were not deemed critical enough to necessitate a thorough SPC analysis. Then we prioritized the remaining metrics with respect to their significances and started to work on high priority metrics as we had a limited time period (metrics 1 through 6 in the Table 4.1). After a more detailed analysis, we decided to disregard Test Performance metric as we could not gather effort data for different tests separately. Similarly, we decided to exclude Requirements Stability metric since the metric was recently defined the number of data points was not sufficient for a thorough analysis. In the end, our scope has been drawn by Rework Percentage, Defect Density, Productivity and Inspection Performance metrics.

Table 4.1 Candidate Measures

No	Measure	Description
1	Rework Percentage	Percentage of rework effort to total effort
2	Inspection Effectiveness	How effective is the inspection process (in terms of defects found per review time)
3	Defect Density	Relative number of defects in a product
4	Productivity	Production amount (product size) per effort
5	Test Performance	Number of defects found per testing effort
6	Requirements Stability	Percentage of added, deleted and changed requirements
7	Customer Support Time	Time passed from receiving an accepted customer problem until corrected code is deployed to target environment
8	Backlog Management Index (BMI)	The amount of trouble report creation rate relative to closure rate
9	Audit Effort Percentage	The amount of audit effort relative to total effort
10	Audit Performance	The number of nonconformities detected during audits relative to total effort
11	Peer Review Effort Index	The amount of peer review effort relative to work product size
12	Total Defect Containment Effectiveness (TDCE)	Number of pre-release defects relative to all defects found in a product
13	Trouble Report Aging	Time elapsed between the initiation and the resolution of a trouble report

For each selected metric, we first performed a detailed analysis in order to establish a good understanding on metric basics and data analysis mechanisms. We then worked on metric data to determine company-specific parameters, to carry out relevant normalization procedures, to utilize relevant SPC techniques and to interpret analysis results. In this way, our study is structured as a single case with embedded units consisting of the mentioned metrics. We used historical data in order to calculate control limits and to draw control charts. The collected raw metric data was not applicable for most of our SPC measures. Therefore we had to make some derivations by using various data from different sources for appropriate SPC analysis. As control chart is one of the most sophisticated data analysis tools within SPC, we demonstrated practical evidence on the utilization of SPC via control charts. We detected outliers and investigated them in order to understand the variation, and tried to figure out whether SPC has been an effective tool to capture assignable causes. Finally we tried to reach a common conclusion about the means and benefits of Statistical Process Control in software domain.

4.2. Case Analysis

After establishing the required theoretical knowledge, we performed the case study for the selected metrics in order to obtain practical evidence on the validity of our reasoning about Statistical Process Control. The data is obtained from seven projects. These projects have various characteristics regarding their sponsors (internal or external), scopes (MIS, real time etc.), and market places (commercial or military). Accordingly, their sizes, schedules, budgets and assigned work forces are different. However, we could not perform separate analyses for different project types as the data was not sufficient.

For confidentiality purposes, the commercial project and CSCI names are hidden. Similarly, the actual data has not been presented on the charts. Before drawing Individuals charts, the data is multiplied by a constant factor. However we still observe the same trend and detect the same outliers after this modification despite the changes in mean and variance values. As the limits of a u-chart are dependent on the size measure, a similar change would cause errors while identifying the outliers. For this reason actual data is used to draw u-charts, but the numbers on

the y-axes are hidden. For each table, the data are recorded in time order so that the limits are correctly calculated.

In some of our measures, we needed information regarding the size of software code. In these analyses, our computation is based on non-comment and non-blank lines of software code. We also excluded test stubs and totally reused code from SLOC count.

During our study, we used Individuals charts and u-charts for analyzing metric data. We also decided to restrict our analysis to the tests of control limits instead of investigating other tests for detecting the outliers. In the following sections, we will explain the details of our study for each metric.

4.2.1 Rework Effort Analysis

In the company, rework is defined as any modification to configuration items after final peer review (see section 4.2.4) of the first release and changes to internal/external baselines. Any change before the product is put into configuration control is regarded as part of development, but not as rework. Such nonconformities are recorded on and tracked through Action Item Lists. On the other hand, Problem Reports and Document Change Requests are used to document defects that are detected after an internal or formal baseline is established. For this reason, any defect recorded on a PR or DCR form is believed to cause rework. Moreover, the total problem resolution time, which corresponds to the rework effort in terms of man-hours, is recorded on these forms. Thus, we can calculate the total rework effort by summing up the efforts on trouble reports within the dates corresponding to related project phases. The timesheet data is also collected daily for each individual and we can obtain effort amounts in project level.

As the rework effort is calculated for more than three years in the company and the results are utilized for process improvement studies, we can assume that there is a firm understanding on the meaning of measurement. However, as the causes of defects are not recorded on the trouble reports, we did not have the opportunity to make an analysis on the rework percentages **related to** different project phases.

Therefore, our study has been limited to measure rework percentages **within** specific time intervals.

A trouble report related to a document includes many defect items on the same form. Nevertheless, the defect fixing (including analysis, correction and relevant reviews) effort is recorded as one measure for all the defects. As different defects on a form would have different origins, we were not able to make an analysis regarding defect origins. In order to perform this analysis, we initially thought to gather rework effort from the timesheet database by counting the effort of an activity related to a previous phase when that activity appears after a future phase has already started. Although this may work in a waterfall model, consequent phases overlap during the lifecycle of incremental and evolutionary developments. Thus, we rejected this idea.

We initially thought to perform SPC analysis both on a monthly basis and for each project phase separately. But we realized that the project phases were overlapping for different CSCIs and builds in the projects. As we did not have the necessary infrastructure to obtain CSCI and build based effort amounts, we were unable to derive exact rework percentages for different project phases. For this reason, we decided to analyze our data in weekly units. We recorded rework effort and total effort amounts for each week for each project.

Meanwhile, we realized that some of the trouble reports stayed open for more than one week (7 days). For such trouble reports, we assumed that the effort amount is uniformly distributed among different work days. Thus, we calculated a weighted effort for the weeks, where the weight is determined by the number of days that the trouble report stayed open in the corresponding week. For instance if the initiation date of a trouble report is 22nd of May - Thursday and closure date is 27th of May - Tuesday, 33.3 % of total effort is counted for week 2, and 66.6 % for week 1 (4 days from Thursday to Sunday; 2 days from Monday to Tuesday).

As the rework amounts increase during inspection and testing periods, the variation with respect to different points in the life cycle can be regarded natural. In order to smooth out the effect of this natural variation, we decided to perform our analysis on a four-week period through which the rework percentage is

assumed to be within certain limits. Therefore, we summed up four consecutive weekly rework and total effort amounts separately and derived rework percentages for each of these four-week periods. Moreover, we separately analyzed documentation and coding rework percentages as they would possess different trend characteristics.

In order to derive the organizational level control limits, we calculated company-wide monthly rework percentage amounts combining all projects' data (Figure 4.1 and Figure 4.2). Moreover, we disregarded the rework data for the periods before implementation phase for coding rework although some coding has been done during throw-away prototyping in some of the projects. Finally we drew Individuals charts to analyze each project separately with the organizational upper and lower control limits. Now we will go over each project one by one:

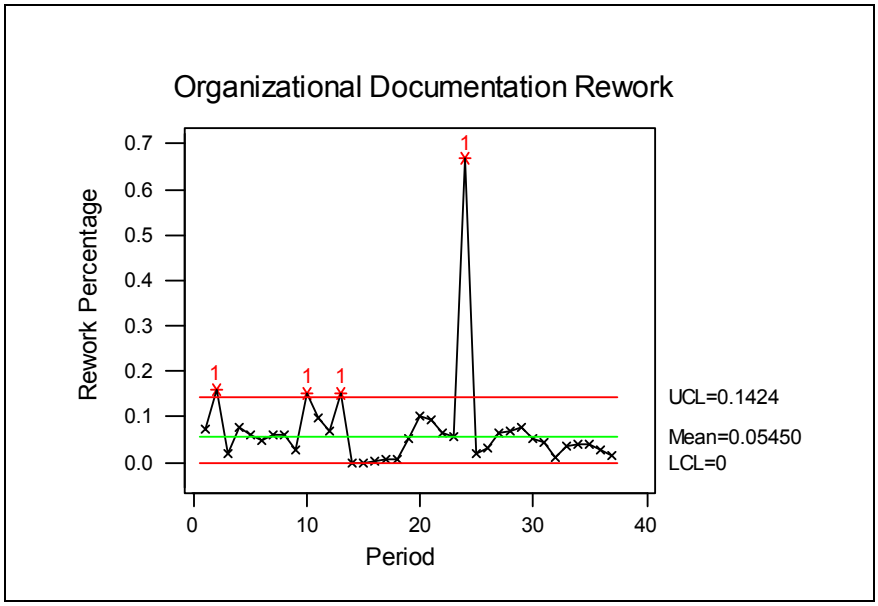

Figure 4.1 Organizational Documentation Rework
(Outlier removed during limit calculation)

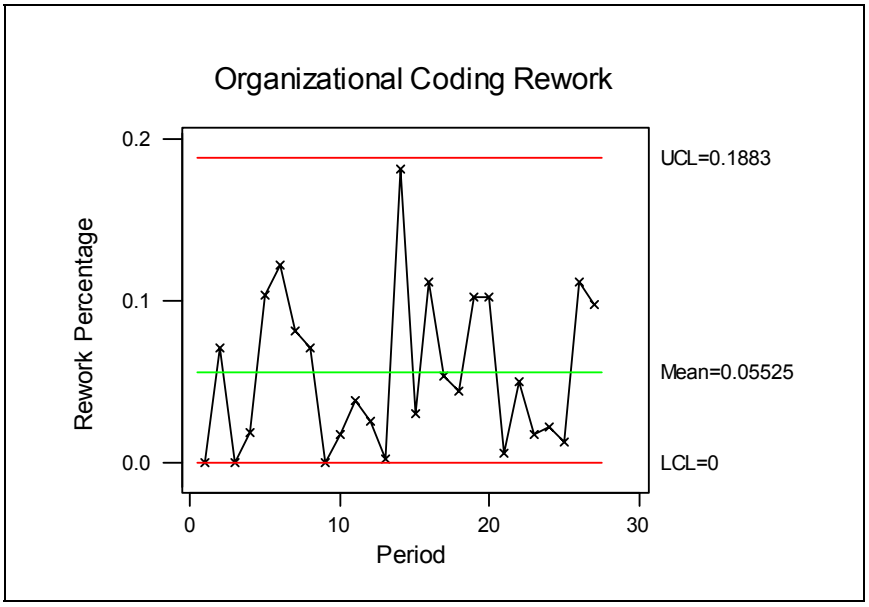

Figure 4.2 Organizational Coding Rework

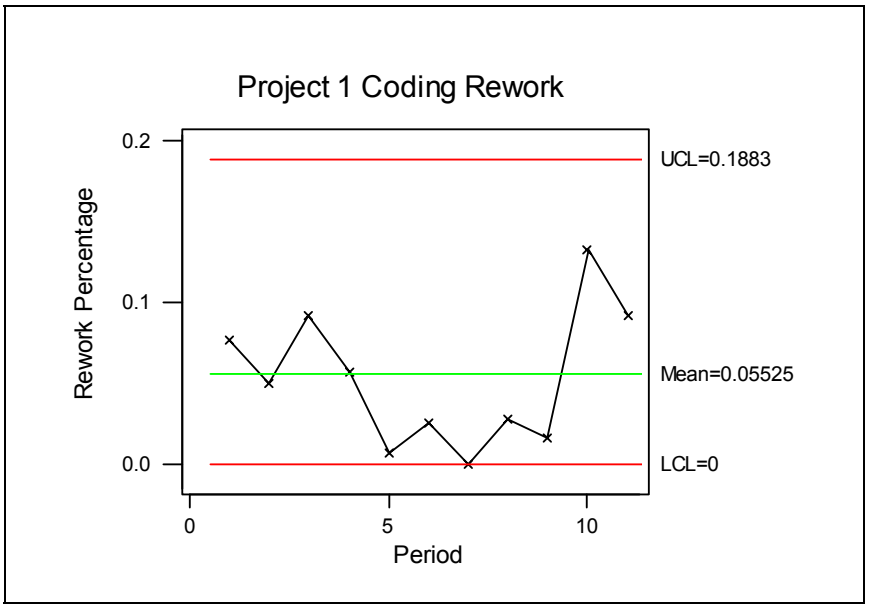

Figure 4.3 Project 1 Coding Rework

Project 1: There is no outlier in Project 1 code rework data (Figure 4.3). However, documentation rework percentage exceeds the upper limits in periods 3 through 6 spanning an interval from 28.01.2002 to 29.04.2002 (Figure 4.4).

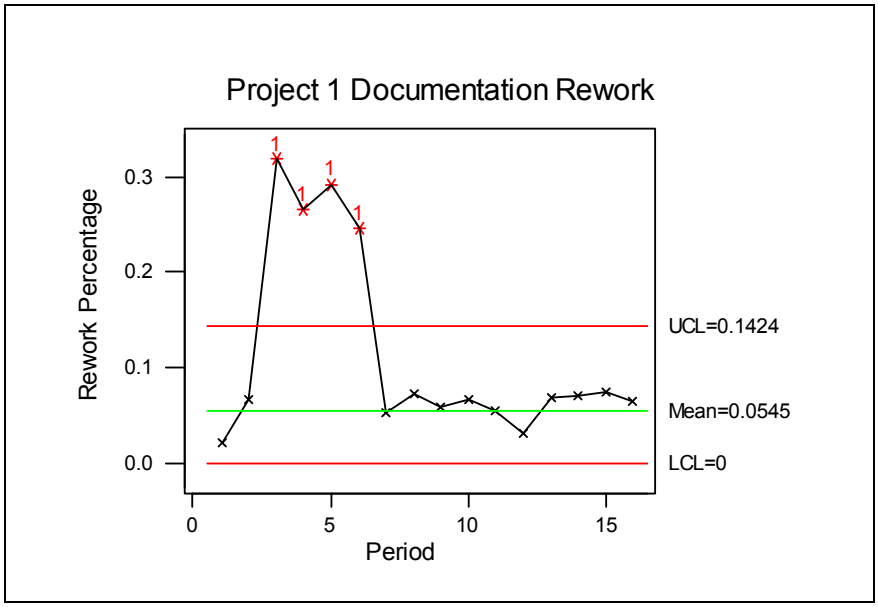

Figure 4.4 Project 1 Documentation Rework

In order to figure out the reason of this happening we talked to the project manager and learned that the project witnessed a radical structural change during this period. The CSCIs are divided into smaller builds in order to release some functionalities in a shorter time, the design documents are modified as a whole to include more detail and accordingly the documentation changed.

Project 2: No out-of-control situation is observed in code and documentation rework data (Figure 4.5 and Figure 4.6).

Project 3: No out-of-control situation is observed in code and documentation rework data (Figure 4.7 and Figure 4.8).

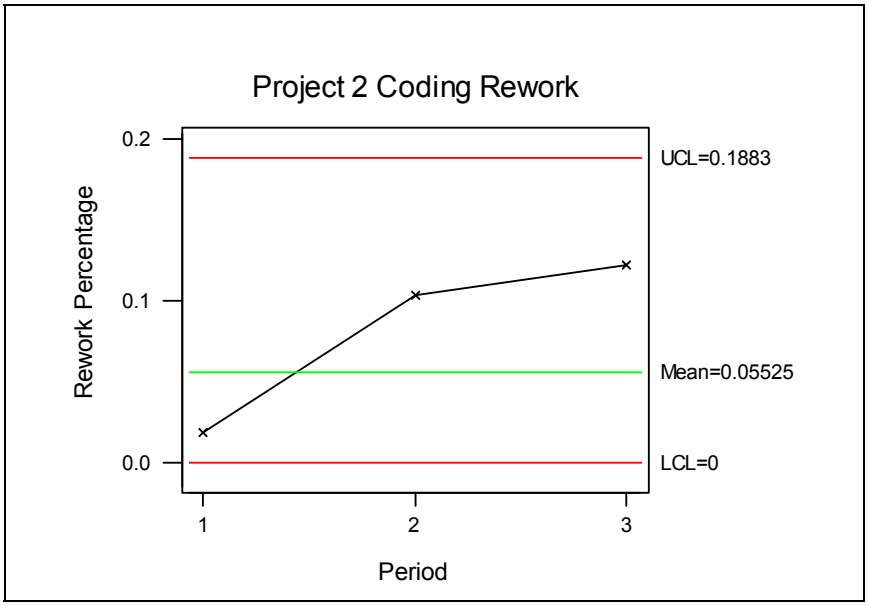

Figure 4.5 Project 2 Coding Rework

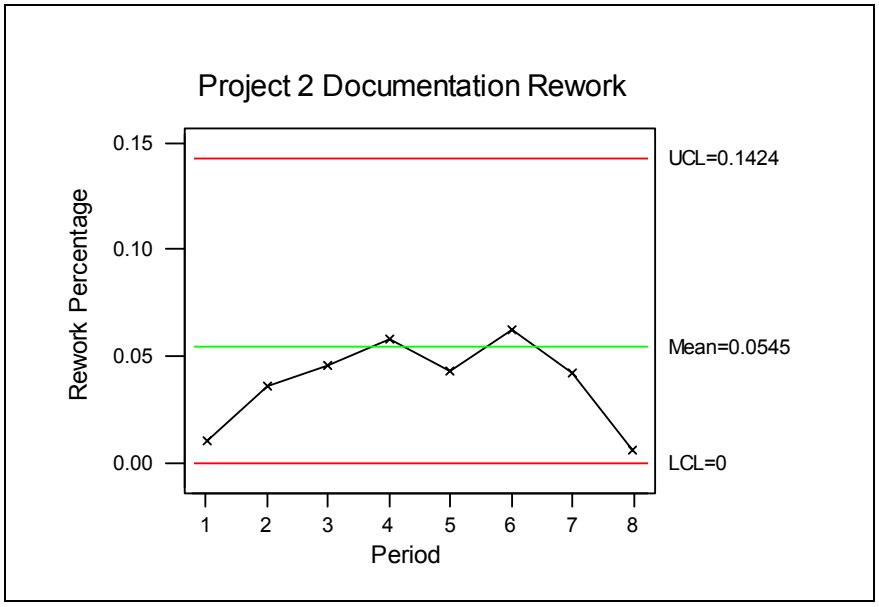

Figure 4.6 Project 2 Documentation Rework

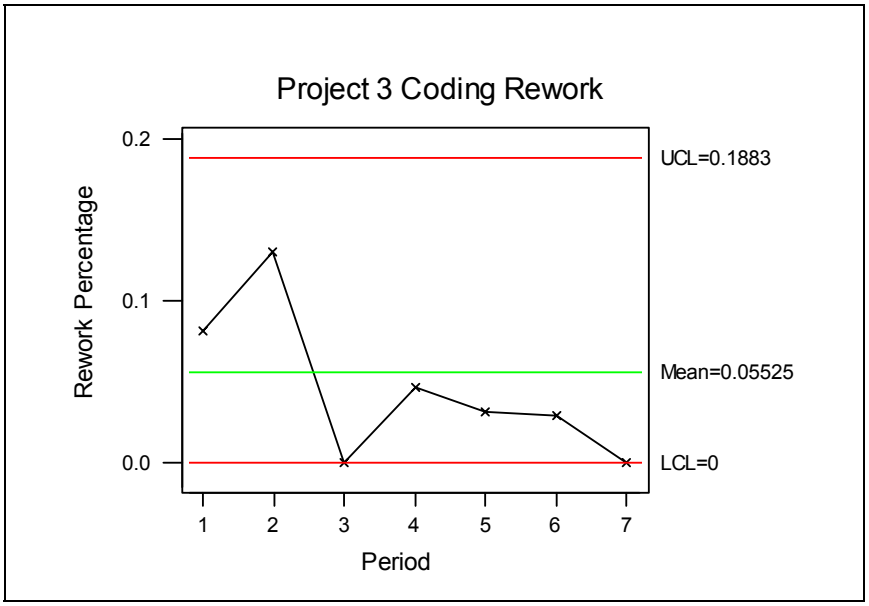

Figure 4.7 Project 3 Coding Rework

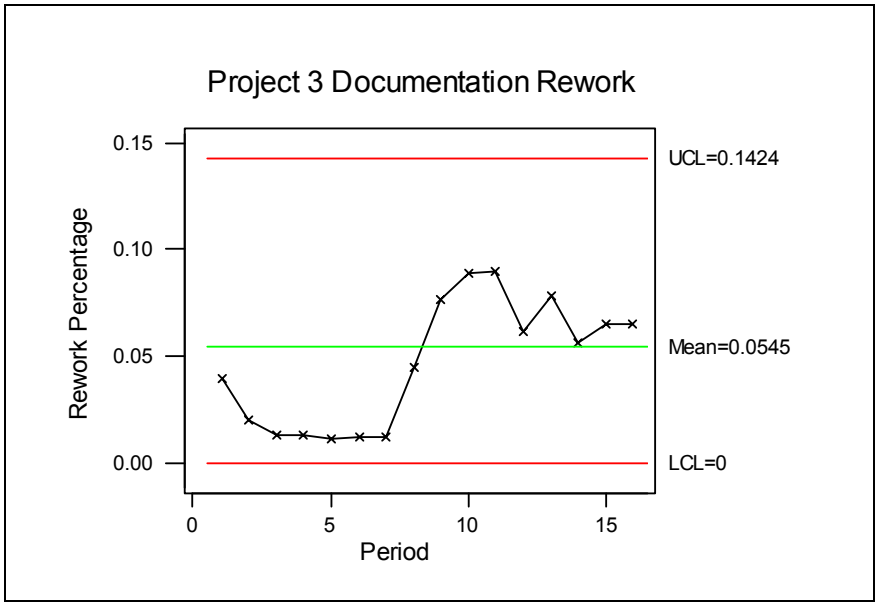

Figure 4.8 Project 3 Documentation Rework

Project 4: No coding has been done yet. However, we detected a long period during which high rework percentages are observed (points 7 through 11 spanning a period from 27.08.2001 to 28.01.2002), two of which are more than upper control limit (Figure 4.9). When we asked to the project manager we understood that there were some important factors causing this occurrence. Firstly, the project was an internal project and it had been suspended a few times with different reasons. In the meantime, the requirements gradually changed because of the need to apply new technological advances and the improved insight on the subject. And during this period, the scope of the project was extremely modified with the approval of top management. Moreover, there was a high turnover rate in the project and a high number of project members were changed before this era. These new project members needed to modify the existing documentation considering the updated requirements according to their perception. With all these findings, we believed that the rework percentage data in this period can be regarded as an outlier.

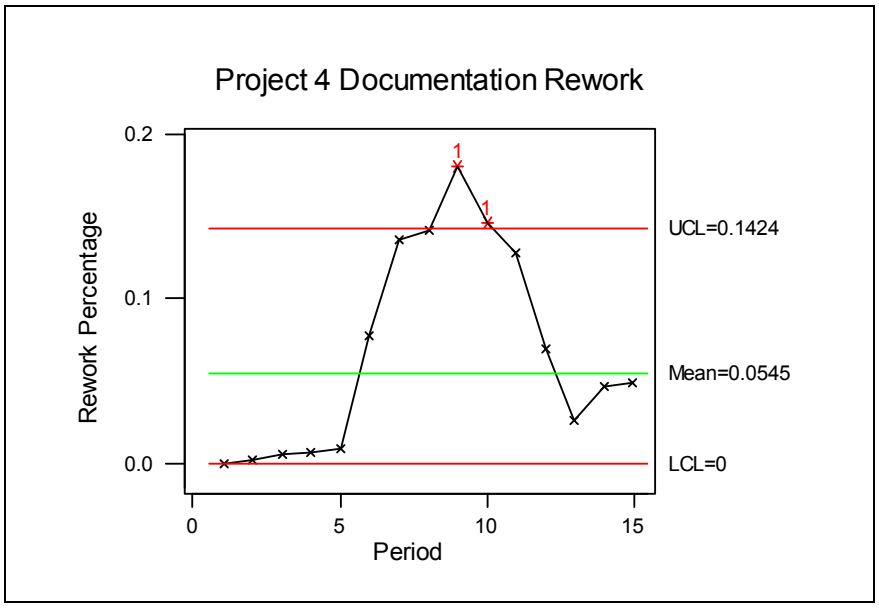

Figure 4.9 Project 4 Documentation Rework

Project 5: There is a strange trend in coding rework percentage data (Figure 4.10). The points 11, 12 and 14 demonstrate high rework amounts covering a period from 27.05.2002 to 29.07.2002 and from 26.08.2002 to 30.09.2002. The chart makes a peak at point 14 and then drops down to normal values. From our project-specific analysis we learned that the customer started to use the product actively after May 2002. The high rework trend after 27th of May is a reason of problems detected by the customer during product usage. The project manager relates this occurrence to the nature of the project and to the behavior of the customer. The customer was a partner firm of our company and this increased their power to act on the requirements. This flexible mood caused the customer to be less concerned about the stability of the requirements and the modifications became unavoidable after product installation. Moreover, the coder was not experienced with the programming language and this caused more rework than expected amount during maintenance phase. As a result, high rework activities are anticipated.

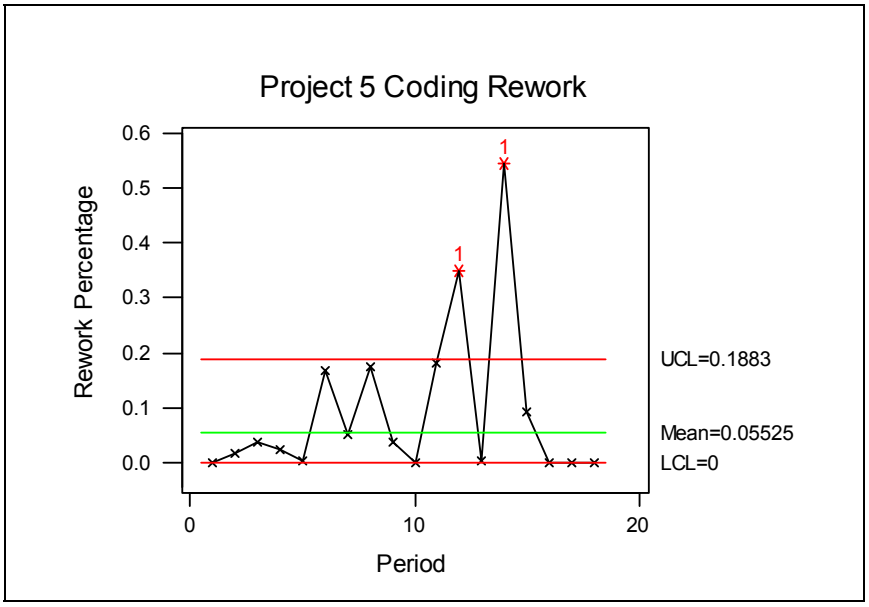

Figure 4.10 Project 5 Coding Rework

When we look at documentation rework chart (Figure 4.11), we do not detect any out-of-control situation. However, point 12 makes a peak between the dates 27.05.2002 and 24.06.2002, showing a parallelism to the coding rework in the same period. This can be interpreted as a natural result of the same changes in the requirements and other related documents.

Project 6: No coding data is available for this project. But the documentation rework process seems to be under control (Figure 4.12).

Project 7: No out-of-control situation is observed in code rework data (Figure 4.13). However, the documentation rework exceeds the upper limits from 24.04.2000 to 26.06.2000 (points 4 and 5 in Figure 4.14). After talking to the project manager, we realized that the customer could not appreciate the actual functioning of initial user interface prototypes since the navigation among different screens was not apparent in the prototypes. Thus, the requirements had to be updated after implementation considering good customer relations and the corresponding rework is visualized in our charts as outliers.

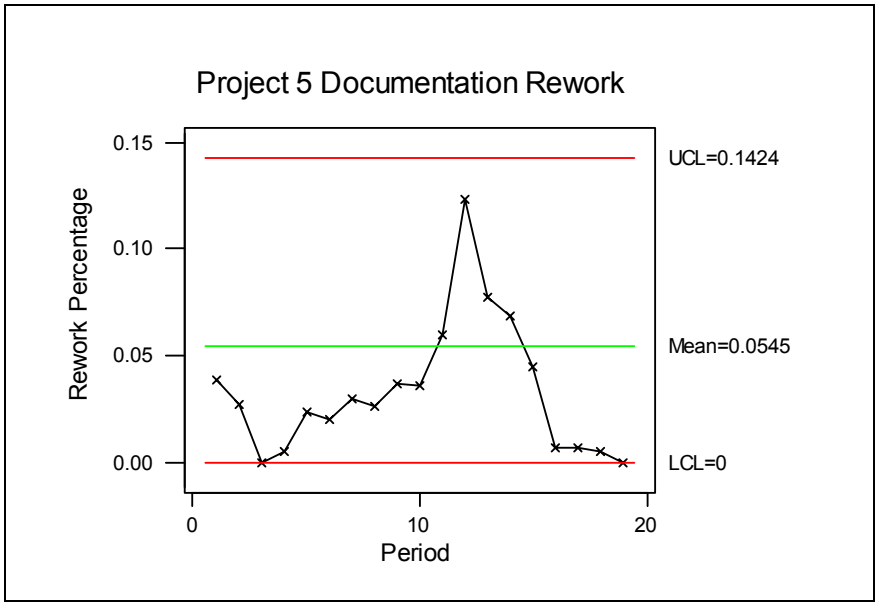

Figure 4.11 Project 5 Documentation Rework

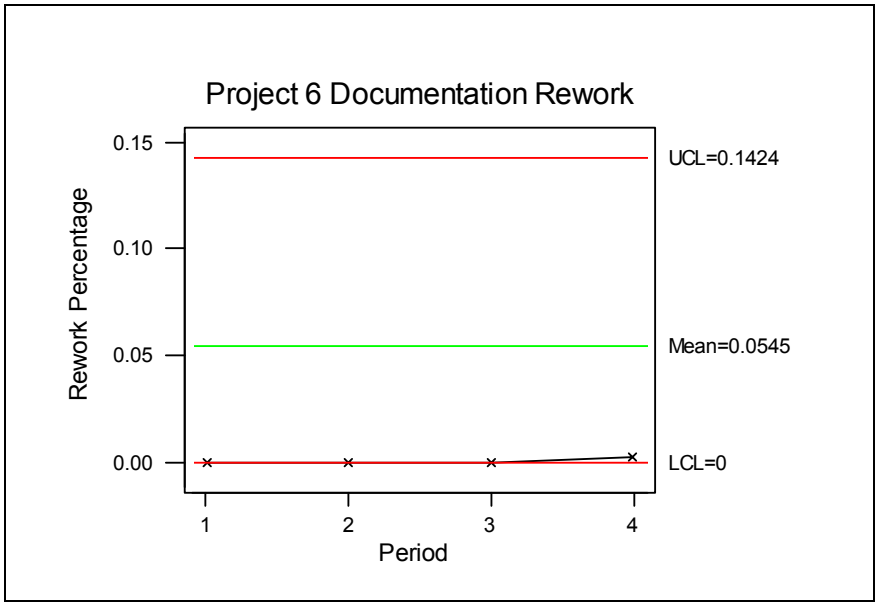

Figure 4.12 Project 6 Documentation Rework

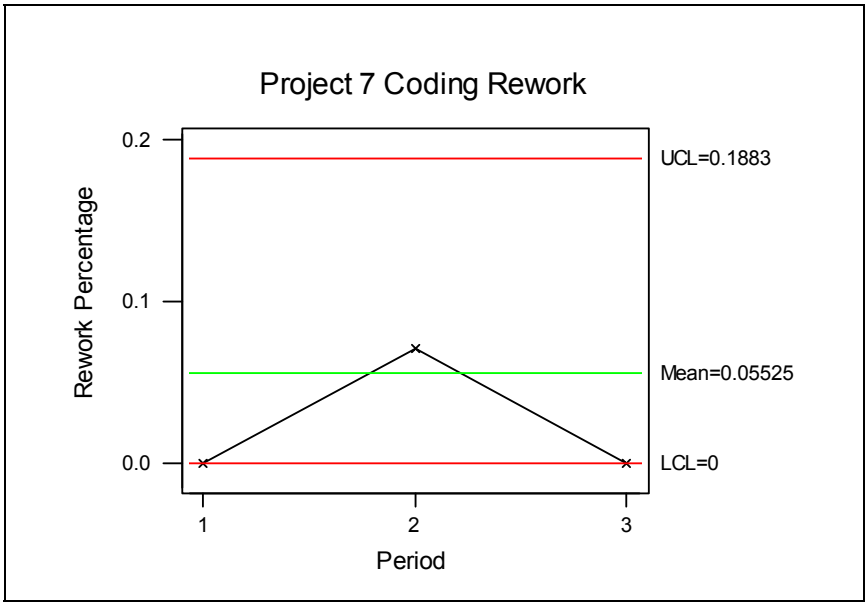

Figure 4.13 Project 7 Coding Rework

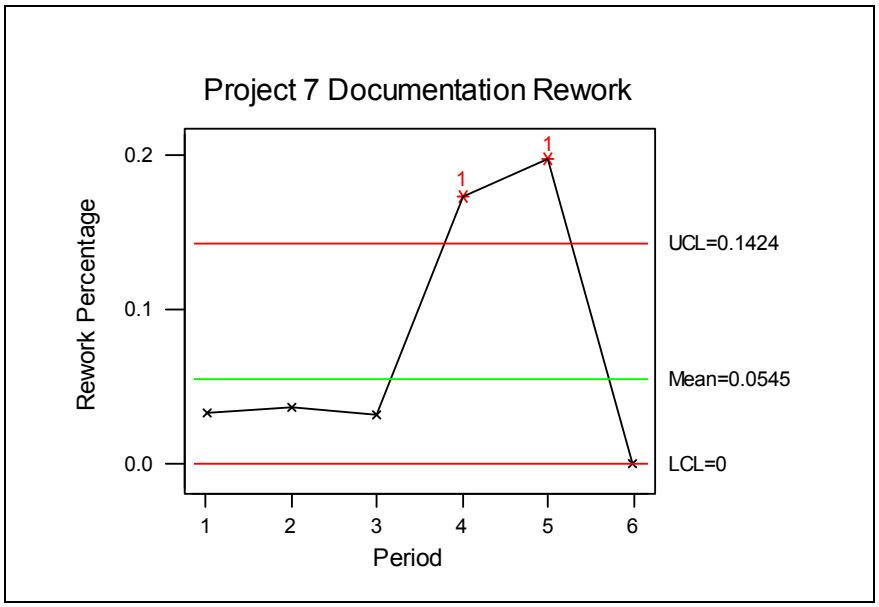

Figure 4.14 Project 7 Documentation Rework

We see that control charts acted as means of visualizing process performances and detect various abnormal situations in projects. As our study has been on historical data and past events, we could only comment on the reasons of process deviations without having a chance to take on-time corrective actions. Nevertheless, such a study provided an opportunity to view some of the mistakes during performing the processes and lead us to think on improvement actions in order to avoid similar situations in the future. From our analysis we deduced that:

- Changes in the structure of a project (the life cycle and architecture) result in high rework amounts.
- Scope changes result in great amounts of rework.
- High turnover rate is an important reason for high rework amounts.
- Internal projects and the projects that are performed to partner organizations are apt to more rework in the later phases. Therefore, special

attention should be devoted to these projects considering the customer's relaxed approach.

- The lack of experience causes high rework.
- User interface prototypes without navigation may mislead the customer and cause more rework in the later phases.

It is obvious that these findings are not some hidden facts or surprising deductions. However, the results prove the success of control charts on detecting some out-of-control situations that cause high rework in the projects. On the one hand, this is an encouraging result as we can more reliably believe in the efficacy of control charts to capture any other deviations in the processes. On the other hand, the charts act as a basis for process improvement as they make a psychological impact on people's minds by showing the deviations in process performance in scientific means. Moreover, on-time action can also be taken as part of overall preventive action activities if the analysis can be performed continuously on ongoing projects.

4.2.2 Defect Density Analysis

The data of all defects found during a review, test, or audit have been collected and tracked through Problem Reports (for code defects) and Document Change Requests (for document defects) since the foundation of the company in 1998. Although the trouble reports evolved within time, the basic defect information such as the subject work product, related project phase, defect priority, initiation and closure date are recorded for all the projects.

Each defect on a trouble report is given a priority, which is classified as low, medium, high, very high and other. After analyzing the data, however, we realized that there was not enough data from each priority category for a successful SPC analysis. Thus we combined 5 priority categories within 3 groups:

Group 1: Combining high and very high

Group 2: For medium

Group 3: Combining low and other

While making this categorization, we made an assumption that we could pay similar attention to the defects in priorities low and other.

The code size is collected for each CSCI in terms of Source Lines of Code, excluding comment lines and blanks. However, as data is not available in software unit level, data points from current few projects did not allow us to perform SPC analysis to code defects. For future studies, it is decided to perform a more detailed SLOC measurement in the upcoming projects. We also thought to use FP count instead of SLOC. However, FP estimation data was not available for all the projects. Therefore, the number of data points was insufficient to perform this analysis.

Considering process control purposes, we decided to restrict our defect density analysis to requirements and design documents and defined size measures as the following.

1. Requirements documents (SRS and IRS): The number of requirements is used to compute size.
2. Design Documents (SDD and IDD): The number of pages is used to compute size.

We believed that the number of requirements is a more meaningful size measure to relate to the number of potential defects for a requirements document. On the other hand, it was difficult to find a similar representative size measure for a design document as its content was dependent on the selected design approach. For this reason, we decided to count number of pages for design documents. As we did not have FP counts for all the projects, we could not perform a complementary study by using Function Points as size measure.

During our study we realized that the number of defects found for a product increases as more and more inspections/tests are performed. At the same time, the product size remains almost the same, which causes defect density to gradually increase from one project phase to the next. As this trend is natural for the measure, our way of analysis should have treated this occurrence as an inherent process property without giving alarms. For this reason, we decided to make separate analyses by comparing the defect density values at the end of each

project phase for two document groups. However, division of data among different priorities, project phases and document groups highly reduced the effectiveness of SPC analysis as the number of samples became insufficient. Therefore, we restricted our analysis for the end of implementation and maintenance phases. As maintenance still continues for some of the projects, the data may need some adjustment at the end of maintenance phases.

The defect information as well as related project phase is recorded on DCR forms. In order to count the number of defects detected in a specific project phase for a specific work product, we summed up all the defects corresponding to that phase from related DCR forms. As some of the defects were recorded with priority N/A in Project 3, we did not include it in our analysis.

The size is computed for each version of the work product and the size of the newest version at the end of the related project phase is used for defect density measurement. However, in some of the projects we realized that different phases occur at the same time because of evolutionary design approaches. In such cases, we used the size of the artifact that is active at the initiation date of the latest DCR written for the related phase.

It is also the case that different builds of a CSCI have the same documentation, with different versions. For instance, an SRS document prepared for build 1 is updated for build 2 with required changes. Therefore, we decided to use the latest version by counting all the defects for previous and current versions. For instance assume that 30 defects are found for 30 requirements (size of an SRS) at the end of requirements analysis phase of build 1. On an ongoing project, we use this data to calculate defect density amount for requirements analysis phase. Let 17 more defects be found since the end of build 2 and the size become 28 (i.e. 3 requirements have been deleted and 1 added). Now we update the first defect density count and use the updated information for the analysis. As the data is also used to establish company-wide performance baseline, we should replace the old data with the new one as soon as build 2 data is collected.

We drew u-charts and Individuals charts for defect density measures of Requirements and Design documents. However, we saw that the u-charts are over-

sensitive for software defect data and capture too many data points outside the limits (see Figure 4.15 and Figure 4.16). Considering the variable behavior of software data, we decided to use only Individuals charts for providing enough flexibility.

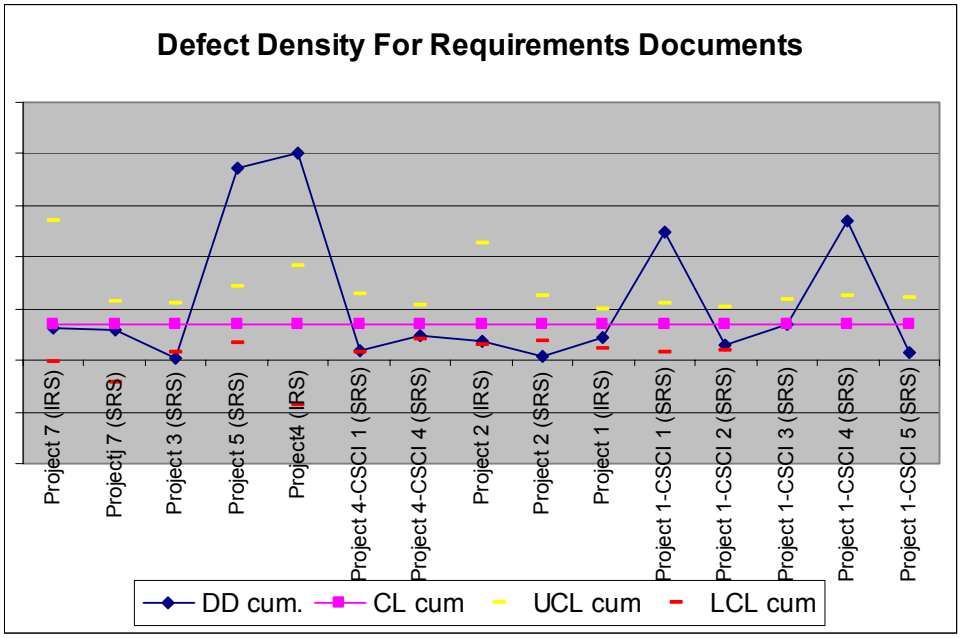

Figure 4.15 u-chart (Defect Density for Requirements Documents)

Requirements Documents: We focused on the defect density amounts of SRS and IRS documents for each priority at the end of implementation and maintenance phases. The initial thing that we captured has been the scarcity of data points for group 1 defects. For this reason, we could not draw separate control charts. Instead, we combined all the defects without considering the priorities so that we could see the influence of group 1 defects to a certain extent.

We also realized a high variability, which made us be suspicious about the stability of the process and the reliability of the results. Nevertheless, we decided

to work on the Individuals charts to see whether we can still obtain beneficial outcomes and to understand the effectiveness of control charts.

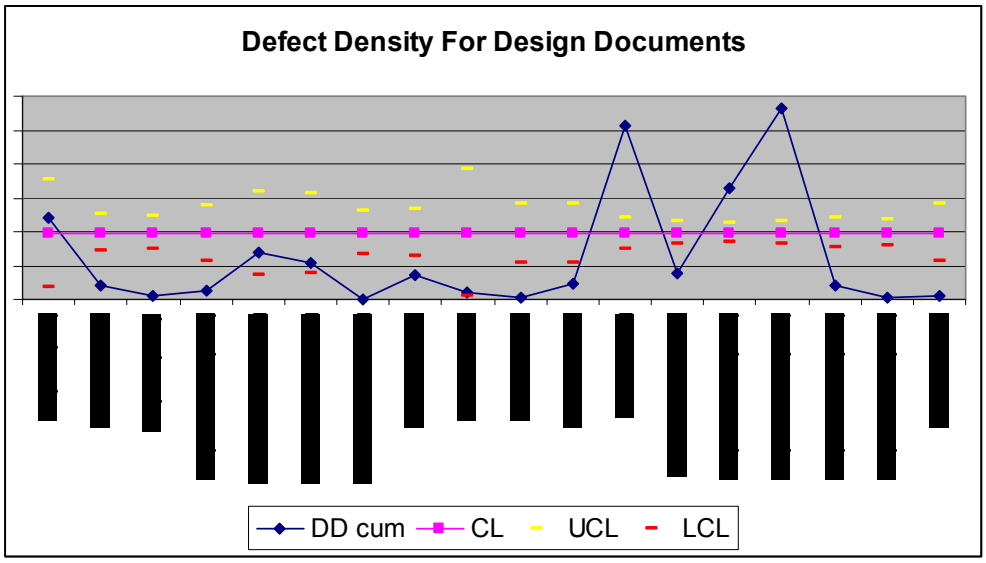

Figure 4.16 u-chart (Defect Density for Design Documents)

Group 2 Implementation: We see that system level IRS of Project 4 (point 4) is very close to the upper control limit on the chart (Figure 4.17). When we talked to the project manager and looked at the IRS document we understood that IRS requirements included interface details regarding all the CSCIs. Consequently, any change to one of the CSCIs affected IRS document and caused updates. Moreover, the project was open to any proposed changes as being an internal research-and-development project. Thus a high defect rate is observed in the IRS document. Assuming this point as an outlier we calculated the control limits again. This time Project 5 SRS (point 3) appeared to be an out-of-limit instance. During rework percentage analysis, we have already recognized an abnormal situation for this project (section 4.2.1). The customer was a partner organization and our company was more tolerable to changes and additional requirements. The

relaxed behavior of the customer during requirements analysis phase, thus resulted in future defects in requirements document. The detection of the same outlier by two independent control charts has been a good reference for the efficacy and reliability of our analysis.

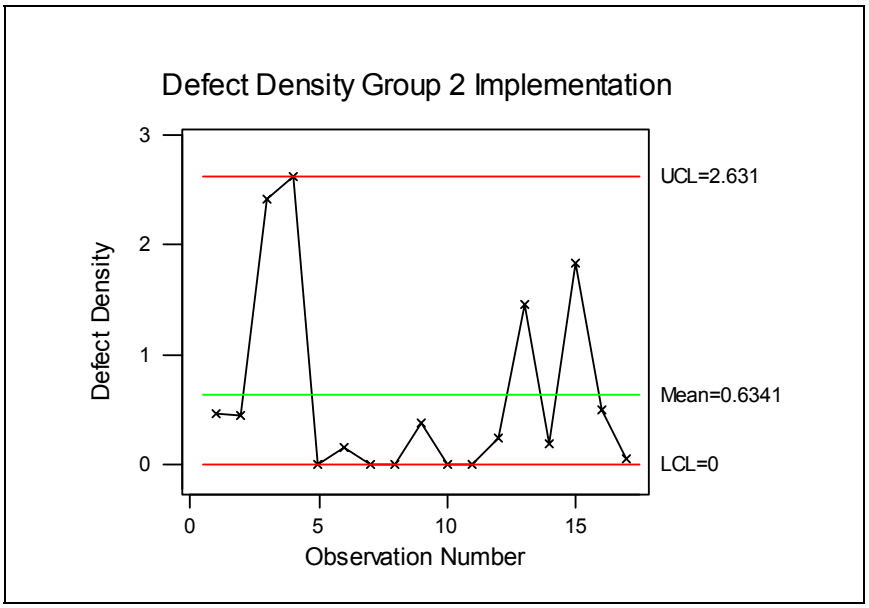

Figure 4.17 Defect Density Group 2 Implementation (Requirements Documents)

Although within the control limits, Project 1-CSCI 1 and Project 1-CSCI 3 (points 13 and 15) generated relatively higher defect density values. The discussions with the project manager and other related project members revealed that the requirements of a future build were added to CSCI 1 and the new requirements in this SRS appeared to be defects as if they were missed during requirements analysis. Similarly, the high defect density of CSCI 3 was a result of a structural change during which two CSCIs were combined to form only one SRS document (CSCI 3 SRS). The added requirements were procedurally recorded as defects even though they were transferred from the other SRS.

Group 2 Maintenance: A very similar pattern is observed as in implementation phase (Figure 4.18).

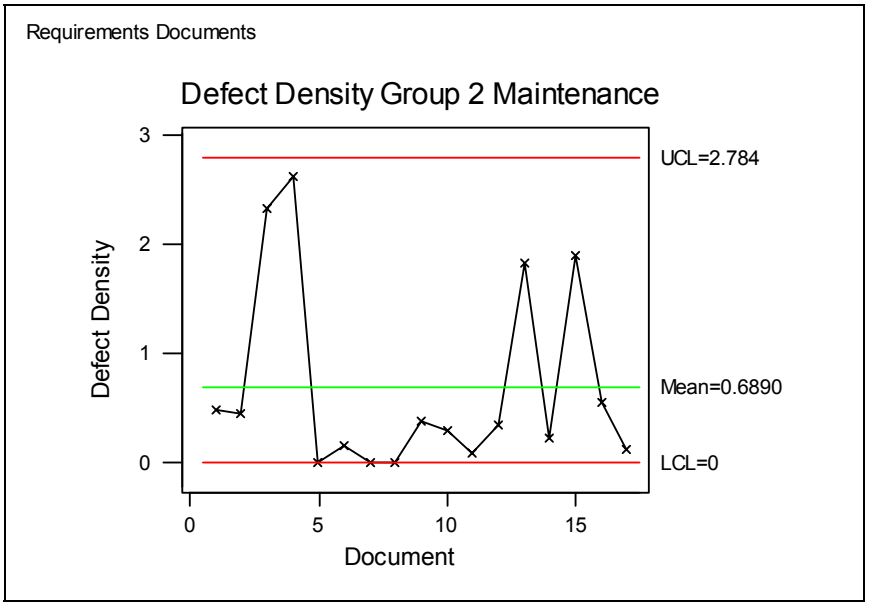

Figure 4.18 Defect Density Group 2 Maintenance (Requirements Documents)

Group 3 Implementation: None of the points are out of the control limits. Group 2 defect density values depict stability (Figure 4.19).

Group 3 Maintenance: Similar to Group 3 implementation data, all points are within the control limits (Figure 4.20).

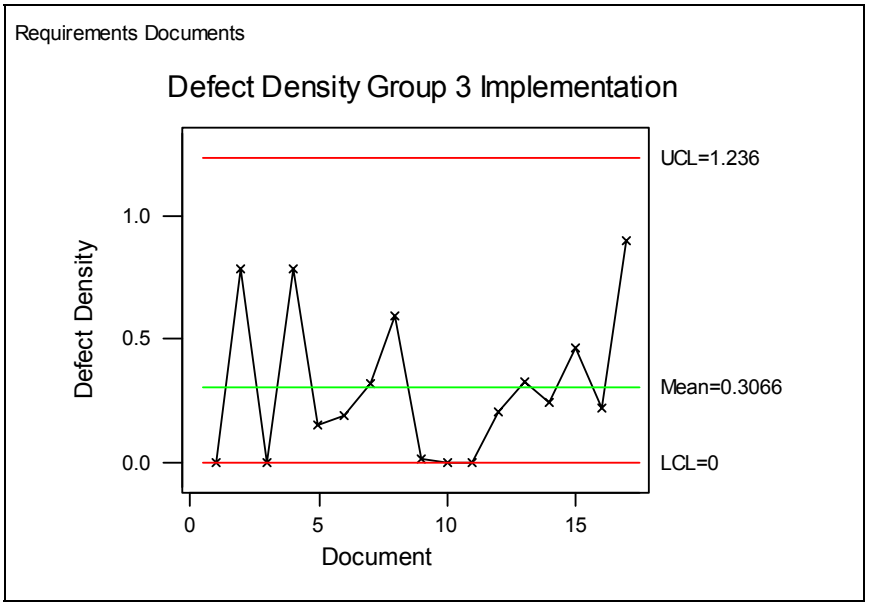

Figure 4.19 Defect Density Priority 3 Implementation (Requirements Documents)

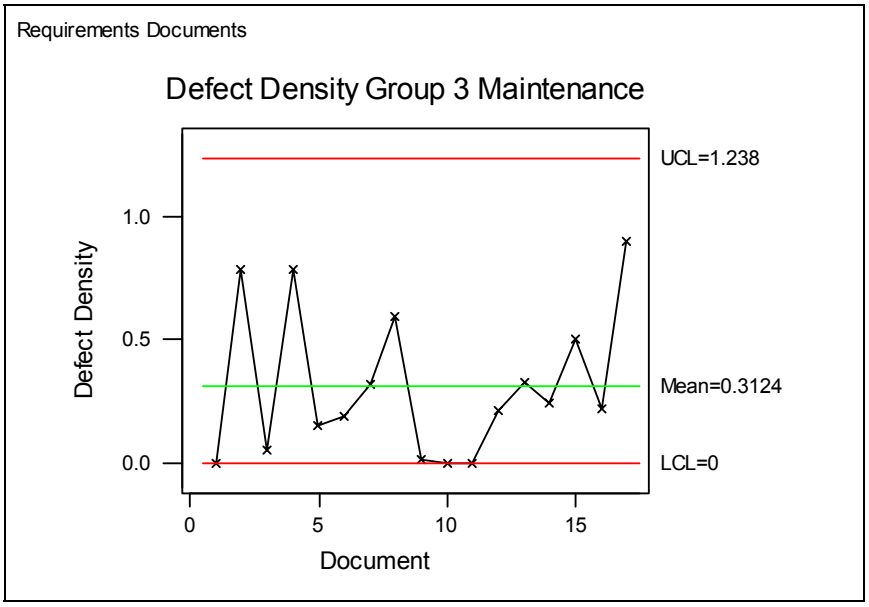

Figure 4.20 Defect Density Group 3 Maintenance (Requirements Documents)

Implementation (All Priorities): We see that Project 4 Software IRS document (Point 5) is high above the upper control limit (Figure 4.21).

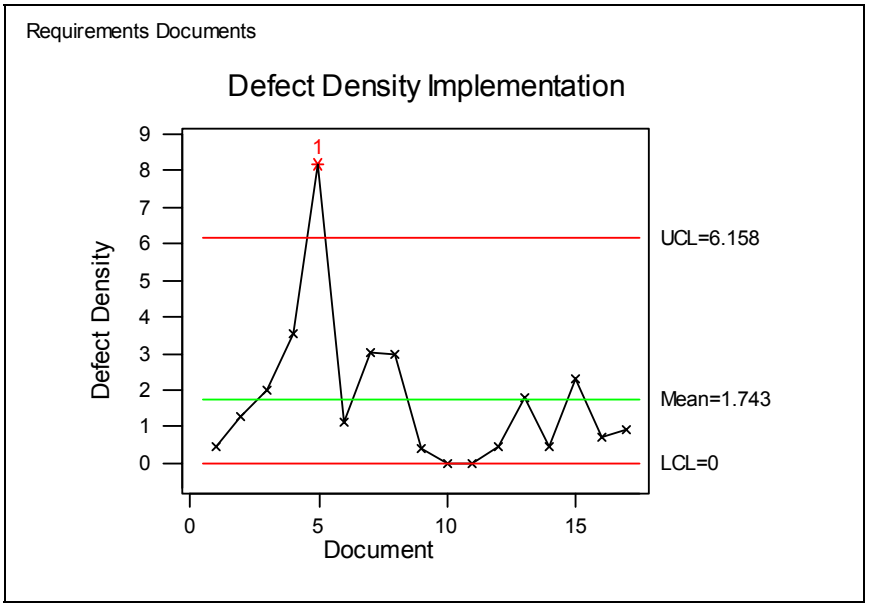

Figure 4.21 Defect Density Implementation (Requirements Documents – All Priorities)

When we investigated the incident we came up against some distinct features about this project. First of all, it was an internal R&D project, which was halt a few times due to personnel requirements in some other external projects. This caused high turnover rates and created an environment for potential modifications during new personnel assignments. Secondly, the project had two IRS documents: One system level IRS, and one software IRS. As software level requirements are defined, a corresponding software IRS is prepared instead of updating the system IRS. Finally, the project was open to many modifications as the management

supported the implementation of new improvements and further dimensions to this internal project. Therefore, some critical changes were performed related to the scope of the project at certain points in the life cycle. The modifications in each CSCI were reflected in system IRS in earlier phases and software IRS in later phases. The resulting effects are observed in the control charts. System IRS showed up in group 2 defect density charts whereas software IRS became an outlier with the existence of group 1 defects related to high-detail software requirements.

Maintenance (All Priorities): A very similar pattern is observed as in implementation phase (Figure 4.22).

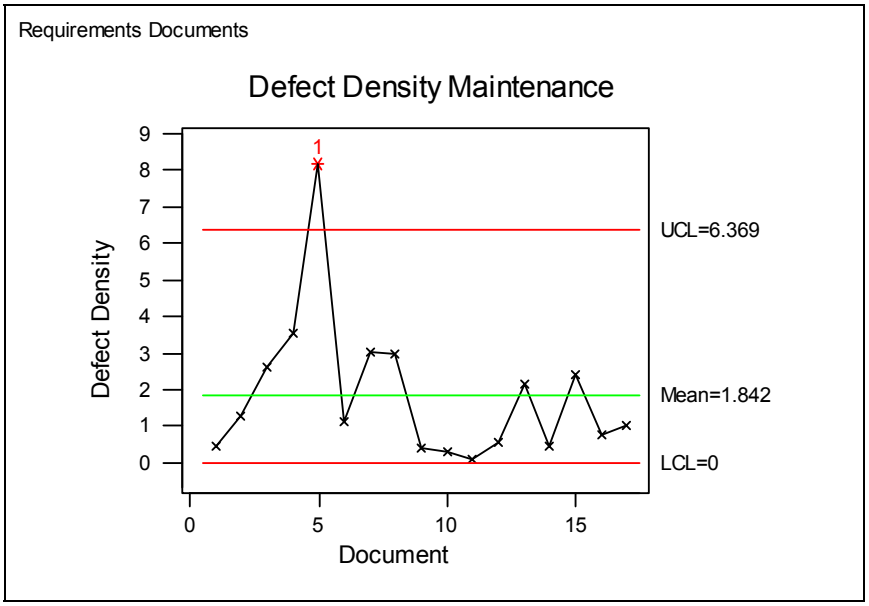

Figure 4.22 Defect Density Maintenance (Requirements Documents - All Priorities)

Design Documents: A similar analysis is performed for SDD and IDD documents. We realized that group 1 defects are seldom and the number of data points is not sufficient for calculating reliable control limits. Moreover, the high data variance causes u-charts to be useless as almost each point becomes an outlier (Figure 4.16). Therefore, we decided to analyze Individuals Charts for interpreting the defect containment amounts of design documents. We drew different charts for group 2 and group 3 defects for the end of implementation and maintenance phases. We also combined all the defects without considering the priorities so that we could include group 1 defects to a certain extent. As the data is collected cumulatively from one phase to the other, the graphs for implementation and maintenance phases looked quite alike.

Group 2 Implementation: Points 11 and 14 are detected as out-of control limits, where point 11 is the IDD of Project 1, and point 14 is the SDD of CSCI 3 in Project 1 (Figure 4.23).

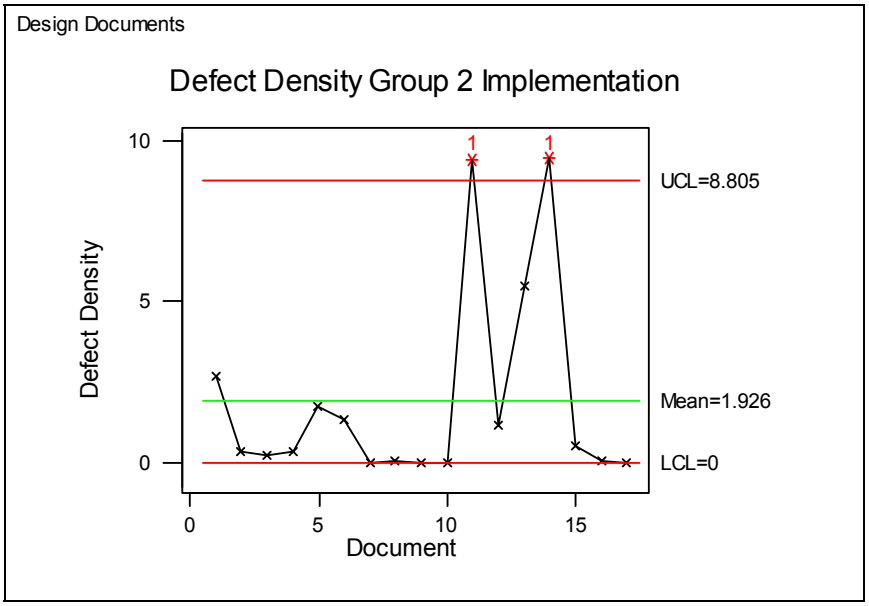

Figure 4.23 Defect Density Group 2 Implementation (Design Documents)

In order to understand the cause of these extremes, we talked to the project manager and other individuals. We realized that there have been structural changes in Project 1 so that the components are divided into different builds. As IDD is a system level document, a change in the design of any of the CSCIs caused modifications in IDD and the resulting picture has been a high defect density value. On the other hand, another CSCI had been added to CSCI 3 a few months ago. This caused many critical additions and deletions to the requirements and the resulting changes are reflected in the design as well. Another distinguishing property of the CSCI has been its size, which became twice as much as the next largest CSCI in the project. Thus the integration needs between subsequent builds became more complex and the products turned out to be more defective. Finally, the staff turnover rate has been very high for this CSCI since the beginning of the project. This is regarded as an important reason for high defect rates in design documents.

There was enough evidence to believe that these two points were outliers. Therefore, we computed the control limits after removing them from the dataset and captured point 13 as another out-of limit point with the updated UCL value. This point corresponds to SDD document of CSCI 2 in Project 1. When we analyzed the specific document, we understood that a change in the backbone overall design of the project mostly affected this CSCI and caused updates.

Group 2 Maintenance: The trend is almost the same as group 2 implementation phase (Figure 4.24). This shows that most of the group 2 defects are discovered before maintenance phase.

Group 3 Implementation: In the chart, the IDD document of Project 7 is represented by point 1, which goes beyond the upper control limit (Figure 4.25).

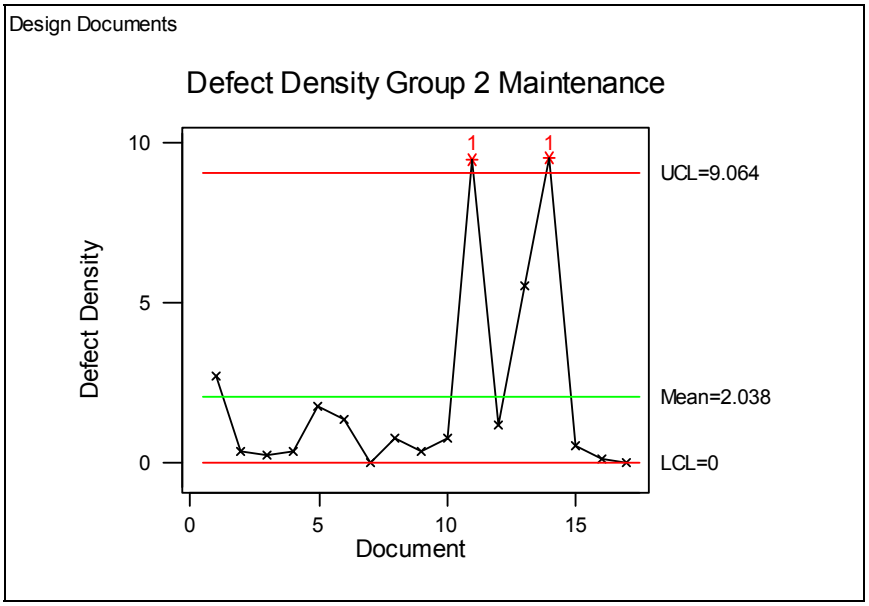

Figure 4.24 Defect Density Group 2 Maintenance (Design Documents)

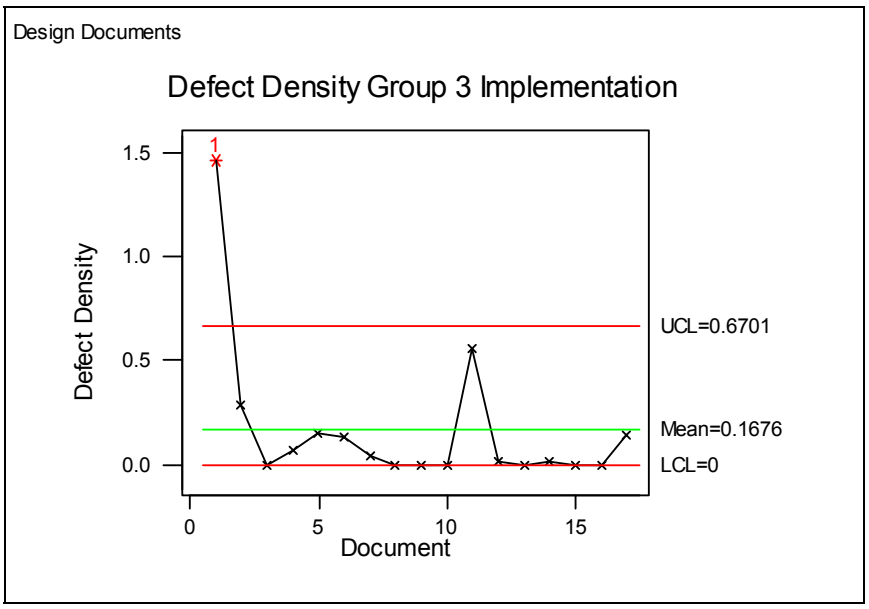

Figure 4.25 Defect Density Group 3 Implementation (Design Documents)

Nevertheless, the project was one of the oldest in the company and none of the project individuals were currently working in the organization. For this reason we could not investigate the reason for this situation. Assuming this point as an outlier, we removed it form the dataset and calculated the limits again. In this case, Project 1 IDD document exceeded the upper control limit (point 11). This inclination of high defect density values for IDD documents made us think that their distribution is different from that of SDD documents. After removing IDD data from our dataset, we obtained another control chart with lower average defect density and upper control limit values (Figure 4.26). In this chart, SDD document of Project 7 became an outlier. With the reason described before, we could not have an opportunity to find evidence about the high defect density value of Project 7 SDD document. We did not perform a separate analysis for IDD documents as the data is not sufficient to calculate reasonable control limits.

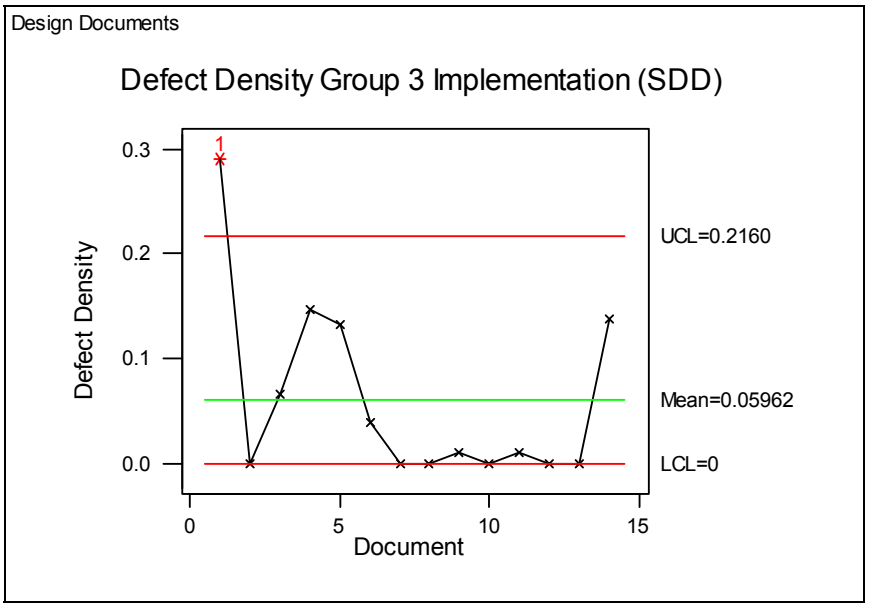

Figure 4.26 Defect Density Group 3 Implementation (Only SDD)

Group 3 Maintenance: We obtained a very similar trend as in Implementation (Figure 4.27). This shows that most of the group 3 defects are discovered before maintenance phase.

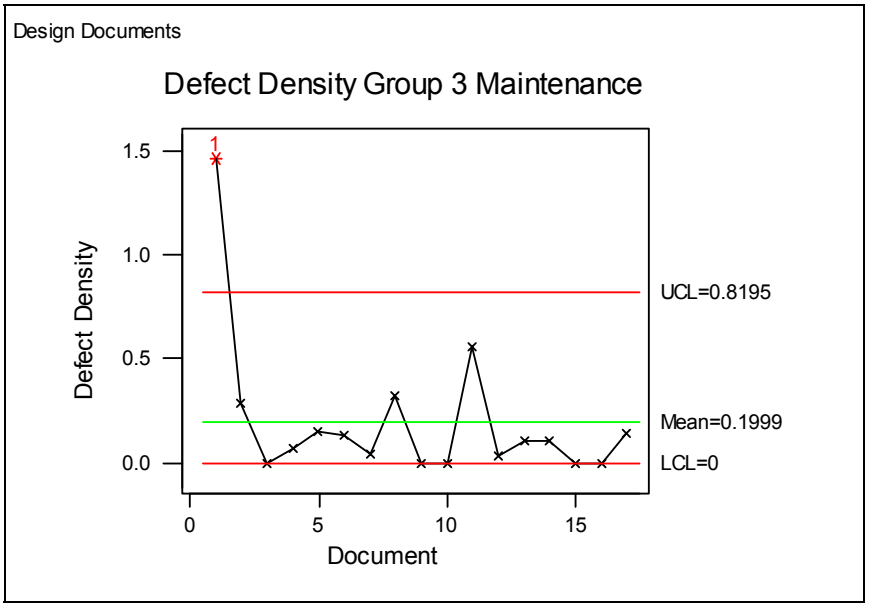

Figure 4.27 Defect Density Group 3 Maintenance (Design Documents)

Implementation (All Priorities): The chart (Figure 4.28) resembles the one that is drawn for group 2 defects at the end of Implementation phase (Figure 4.23). The out-of-control points are also the same. This indicates that a high percentage of defects are group 2.

Maintenance (All Priorities): The trend is almost the same as in Implementation phase (Figure 4.29). This shows that most of the defects are discovered before maintenance phase.

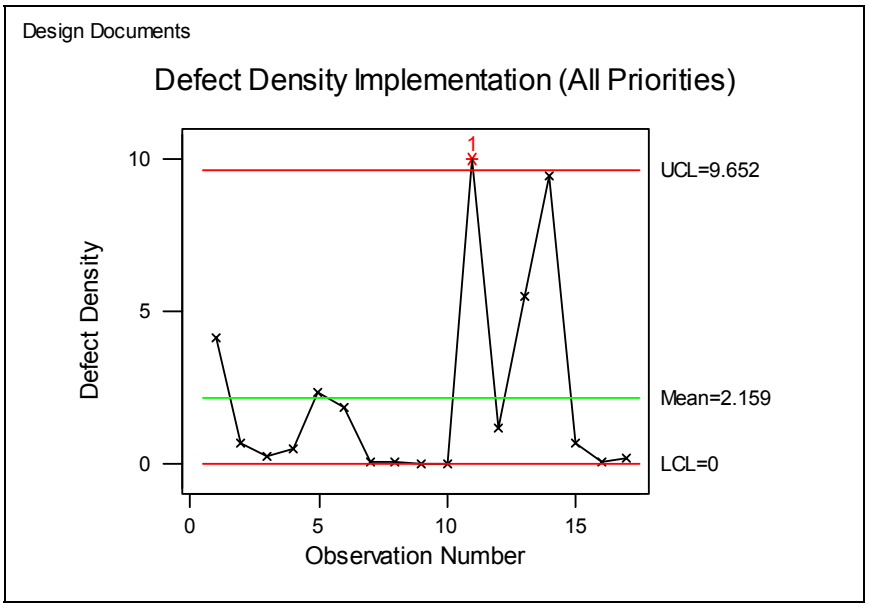

Figure 4.28 Defect Density Implementation (All Priorities - Design Documents)

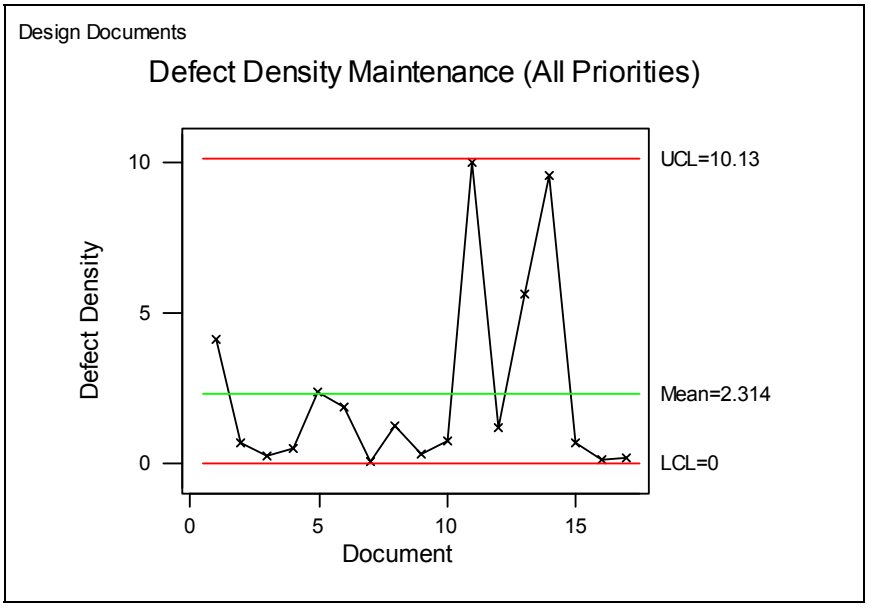

Figure 4.29 Defect Density Maintenance (All Priorities - Design Documents)

Our study revealed some important concerns about defect density analysis. First of all, we observed a very high variability among defect density values of different documents. Although we did not draw the graphs, our findings are similar for code defects. Considering the fact that the processes are well defined and defects are correctly recorded, this incidence indicates that the nature of software processes avoids obtaining stability as in manufacturing applications. This high variability also causes control limits to be too wide, and reduces the sensitivity of control charts.

Nevertheless, the control charts gave us an understanding about the status of different products allowing us to make a comparison with respect to control limits. We could statistically evaluate the difference between the distributions of SDD and IDD documents. We also captured extreme points that demonstrated bad performances of product quality. By thinking on the reasons of these outliers, we prepared a list of lessons learned which provided a baseline for future improvement studies.

4.2.3 Productivity Analysis

Productivity metric has been tracked in the company for more than 3 years. The metric is considered an important one as the top management had an eye on the projects' productivity values. With this idea in mind, we started our study by first defining our constraints, strengths and goals.

Function Point estimation is performed for each CSCI in the company. However, the software development is performed by additional builds in some of the projects and we could not analyze individual builds since FP count is not estimated separately for each build. Moreover, activity based effort data is collected via timesheets; but CSCI level effort data is not available in the company. Therefore, we could not make a productivity analysis for different components.

Because of the mentioned constraints, we decided to measure productivity data in project level. As productivity is the output produced per input, we initially tried to figure out the outputs during software development. The documentation such as

SRS, SDD, SSDD, test descriptions and plans all constituted the work products in a project. However, these documents were prepared separately for each CSCI and we did not have available effort data in CSCI level.

On the other hand, we believed that SLOC count may not give a representative quantification for the projects phases but for implementation. As we wanted to track productivities for each project phase, we regarded FP as a better measure for the whole project size. Consequently, our analysis turned out to be productivity for each project phase using Function Point count as a representative size measure:

Productivity = (Function Points) / (Total effort spent in project phase)

Our measurement depends on the assumption that the project size, in terms of function points, is directly proportional to the effort expended in any project phase. Thus, the measure gave us an understanding about the value of phase effort with regard to the project size. As FP count is recalculated after each project phase, we used the latest measure in our analysis. We restricted our study within four project phases, namely Requirements Analysis, Design, Implementation and Testing.

Effort values are gathered from the timesheet database and productivity values are calculated for each project phase. Nevertheless, we only had 7 projects and only 6 of them had function point results. Thus the available data points did not allow us to compute reliable control limits.

Although we could not draw control charts, we defined how to perform SPC analysis for productivity data. We also realized that the effort measurements and function point estimations should be collected in more detail.

4.2.4 Peer Review Performance Analysis

In the company, the inspection process is carried out by independent reviewers and a summary report is prepared at the end of each review. The critical documents such as SRS, SDD, IRS, IDD and code are inspected by peer reviews. The summary report for a peer review is quite detailed as it includes the names of

the reviewers, their review and preparation times, the number of problems found and review results.

It is possible to investigate peer reviews in three categories:

Draft Peer Review: Performed as a first check before the product is ready for final peer review. The aim of performing this early review is to minimize the number of defects before product is submitted to the customer.

Final Peer Review: This is performed just before the release in order to verify the appropriateness of the product. If no defects are found during this review, the product is released.

Change Peer Review: This is performed to verify the appropriateness of the proposed changes to a product that is previously released.

We see that each review category is performed at different points in the product life cycle. As some of the defects are fixed during a draft peer review, we expect the product to be less defective before entering the final peer review (assuming that bad fixes constitute a small percentage). Therefore, the reviewers will most probably find fewer defects during a final in contrast to draft peer reviews, and the review effectiveness will seem to be worse. On the other hand, different trouble reports are used in order to record the defects that are found during draft and final peer reviews. As the defects found during a draft review are not categorized with respect to their priorities, we decided to perform separate analyses for draft and final peer reviews.

Similarly, the reviewers will focus on the changed sections rather than the whole product when the review is performed to verify the changes. Although the effort will also be low, we do not have enough evidence to assume that the decrease in the number of defects found is proportional to the decrease in the effort. Moreover, the reviewers will not find even a single defect in most of change peer reviews. Therefore, we will not be able to judge a change peer review as ineffective although no defect has been found. As our aim for using this measure is not to evaluate the product but the review, we decided not to include change peer reviews in our analysis.

The reviews are performed by independent individuals within the project. As soon as the independence is provided, any available and skilled personnel may be included in the review team. This structure makes it irrelevant to restrict our analysis to CSCI level since the performance of a specific review will not be indicative for the CSCI for which the review is performed. Thus we decided to work on individual review data within the whole project.

As the structures and review orientations are different between code and documents, it may be natural to observe different distributions for review effectiveness of these products. For this reason, we decided to analyze reviews for code and different document types separately.

In light of these decisions, we put the review data in time order for each review type / product combination. Then we calculated the review effectiveness values by dividing the number of defects by review time (minutes) for each review. In order to investigate whether the metric is statistically related to the number of reviewers, we made a regression analysis. However, most of the reviews were performed by only one reviewer and the remaining data could not capture any relationship.

We drew Individuals Charts for SDD, SRS, UITD, UTD documents and for code. We obtained the following results:

SDD Review Performance: Review 1, which belongs to Project 4, goes out-of upper control limit for final peer reviews. When we investigate the reasoning we see that the reviewer is a very experienced person in the subject. Very small number of defects in the following reviews is also a strong indicator for the effectiveness of first review. We can deduce that most of the potential defects in SDD have been found during review 1. This finding is a good evidence for the importance of reviewers on the effectiveness of review process.

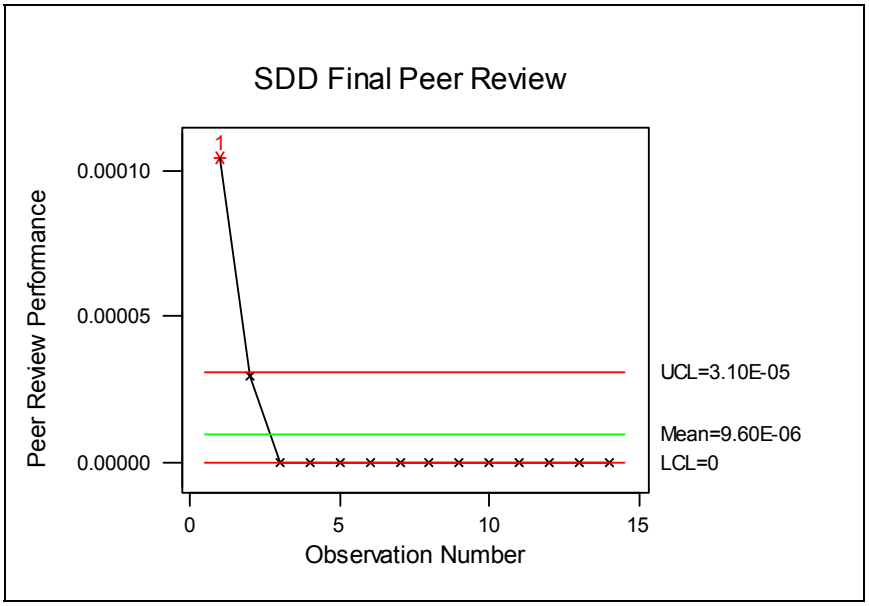

Figure 4.30 SDD Final Peer Review Individuals Chart

If we look at draft peer reviews, we recognize one out of control point (point 20) that represents SDD document of CSCI 3 in Project 1. After analyzing the corresponding review summary report and DCR form with related project individuals, we realized some critical issues about this CSCI. First of all, two CSCIs are combined into 1 within CSCI 3 a few months ago. This major change caused many updates in the requirements as well as in the design. The defect density chart also shows a similar peak for this CSCI and supports our deduction (point 14 in Figure 4.23). Moreover, this update caused the CSCI to be quite big and this caused critical modifications during the implementation of subsequent builds. Finally, the design is found to be inferior than the other CSCIs' and many changes are regarded significant defect removals. Many reasons are supposed for this trend and one of the most important reasons is believed to be high staff turnover rate.

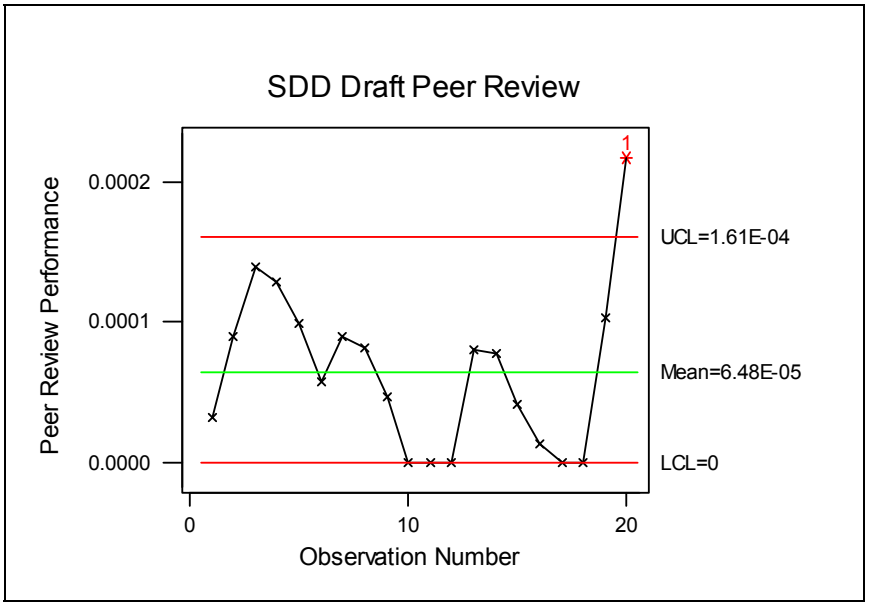

Figure 4.31 SDD Draft Peer Review Individuals Chart

With these findings, we conclude that the reason for the peak in the chart is due to the existence of too many defects in the design of CSCI 3. Considering the extreme conditions for this CSCI, we regarded this point as an outlier. When we removed it from the dataset, point 3 (Project 4 SDD) appeared to be very close to new upper control limit. In order to get some idea about this occurrence, we investigated the case and discovered that most of the recorded defect items in this review are grammatical and formatting errors. We compared some other DCRs in order to understand whether such errors are recorded and corrected in the same detail. We saw that, the detection and maintenance of these kinds of errors mostly depended on the perception of the reviewer. Some meticulous reviewers tried to capture every single detail whereas some were more relaxed. As the detection of these kinds of errors was not very time consuming, the review effectiveness metric showed up to be high.

UTD/UITD Review Performance: Looking at Individuals charts, we detect five out-of limit points (all belong to Project 1) (Figure 4.32) among UTD final peer

reviews and one point (from Project 1) among UTD draft peer reviews (Figure 4.34). As an experimental approach, we removed these points to see the trend of the remaining data and detected 9 more points exceeding the upper control limit (Figure 4.33). In order to understand the reasons for the high review performances, we analyzed related review summary reports, DCR and AIL forms. We first realized that most of the defects are due to formatting and grammatical errors. With this finding, we focused on the reviewers and concluded that some specific reviewers, who had more managerial positions in the project, were more apt to find non-technical defects during reviews. On the other hand, the defects related to technical issues were mostly found for more complex software units. Considering the algorithmic logic within UTD documents, it is reasonable to believe that complexity brings further errors. One final observation we did was the higher final peer review performances achieved by the reviewers who also took part in the previous draft peer reviews. As they worked on the document before, they were more efficient in the second review.

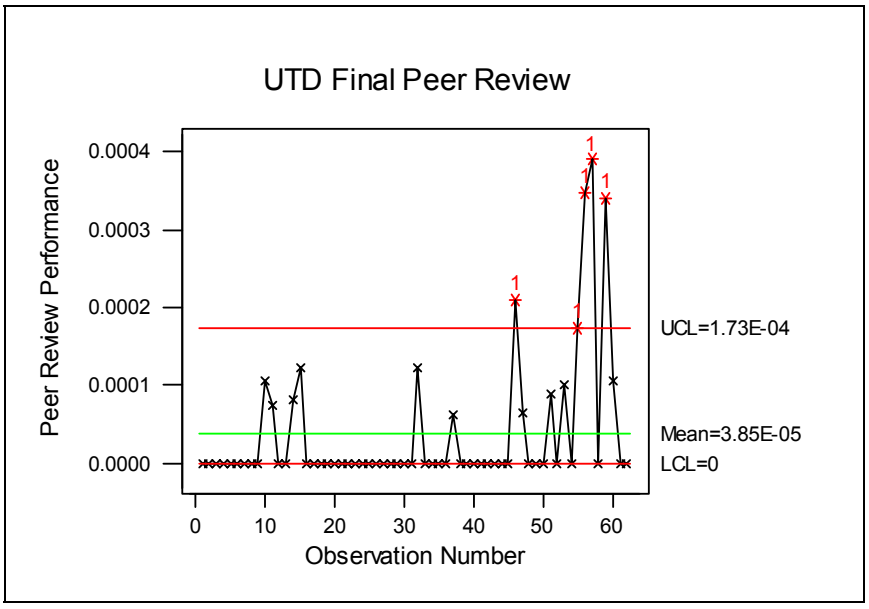

Figure 4.32 UTD Final Peer Review Individuals Chart

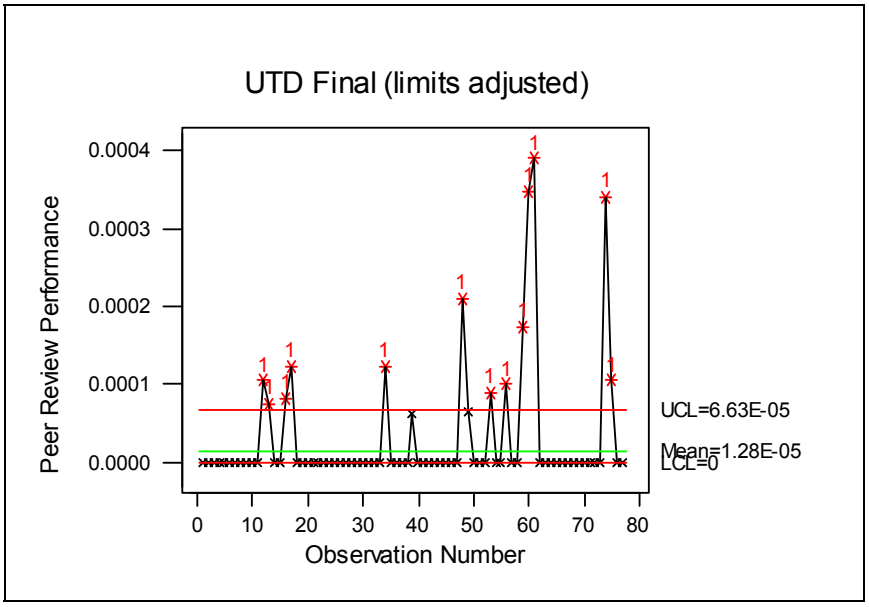

Figure 4.33 UTD Final Peer Review Individuals Chart (Limits Adjusted)

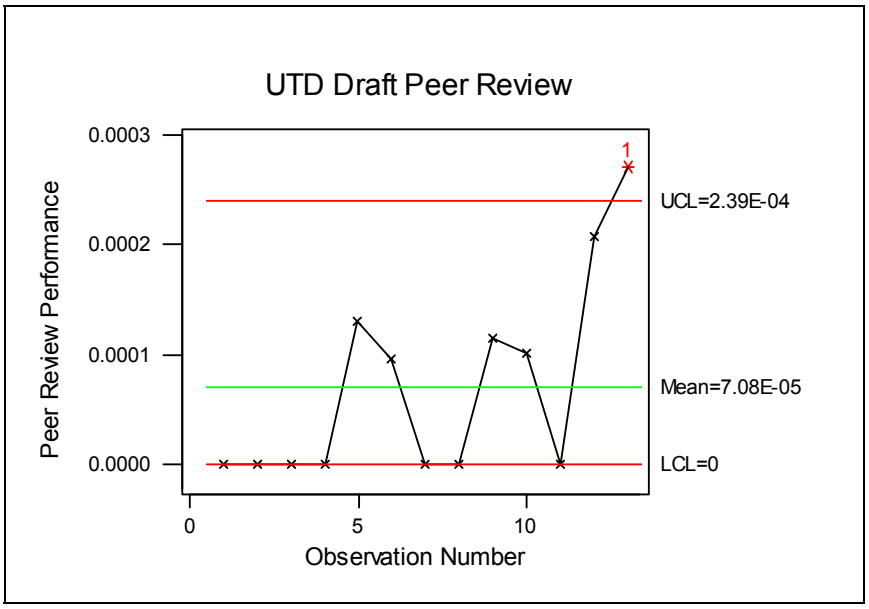

Figure 4.34 UTD Draft Peer Review Individuals Chart

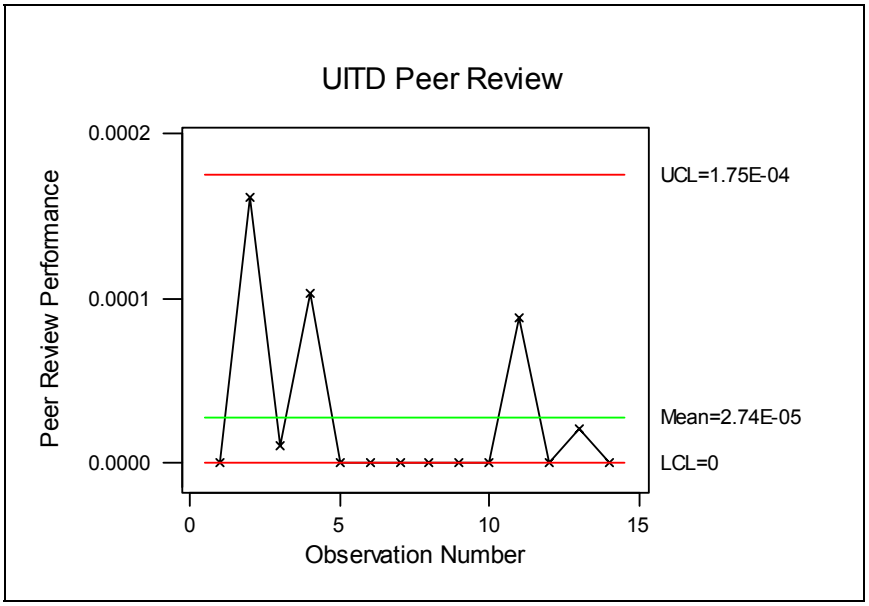

Figure 4.35 UITD Peer Review Individuals Chart (Draft + Final)

As there were not enough data points, we had to combine draft and final peer review data in one Individuals chart for UITD peer reviews (Figure 4.35). As seen in the graph, no point is outside the limits.

SRS Review Performance: Even though none of the points is outside the limits (Figure 4.36), we decided to analyze point 11 as it is very close to the upper control line. We saw that most of the defects were found by one individual in managerial position and were related to non-technical issues. This finding gave us another evidence for the influence of reviewer on the review effectiveness and on the characteristics of the defects found.

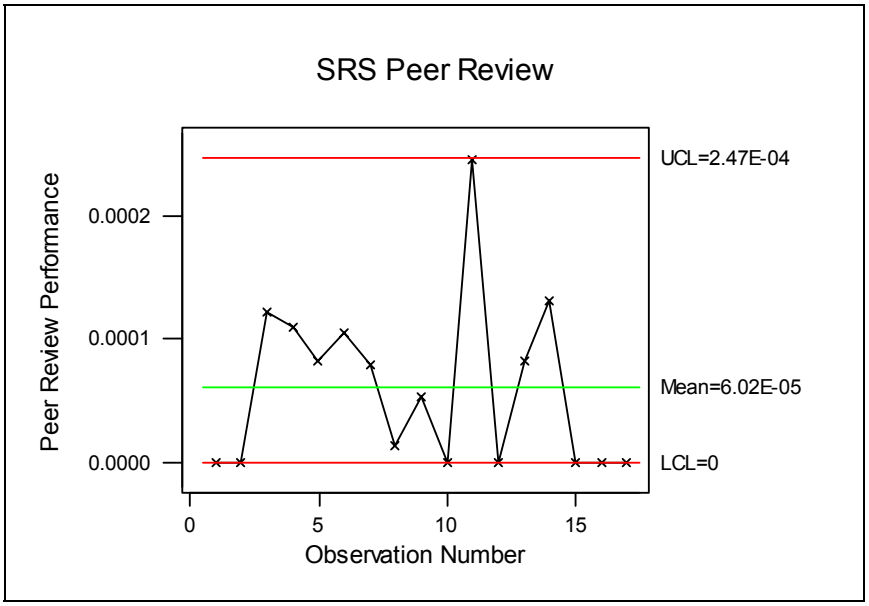

Figure 4.36 SRS Peer Review Individuals Chart (Draft + Final)

Code Review Performance: In the company, draft peer reviews are not mandatory for code reviews and we had only one draft peer review data. For this reason, the Individuals chart (Figure 4.37) represents the final code review performance in the projects. We recognize 5 out-of-limit instances belonging to Project 1. This is not surprising as 96 out of 103 data points are from Project 1. The results in the analysis may be biased because of the scarcity of different projects in the analysis. Nevertheless, the analysis will represent Project 1 code review performance with fewer nuisances. In order to investigate the reasoning behind high review performance values for the 5 points, we looked at review summary reports and talked to the related project individuals. The most significant point we realized was the kind of defect items found during the reviews. The defects found during these reviews were very common in the previous reviews. Therefore, reviewers were inclined to search for the same defects and could find them in a short time.

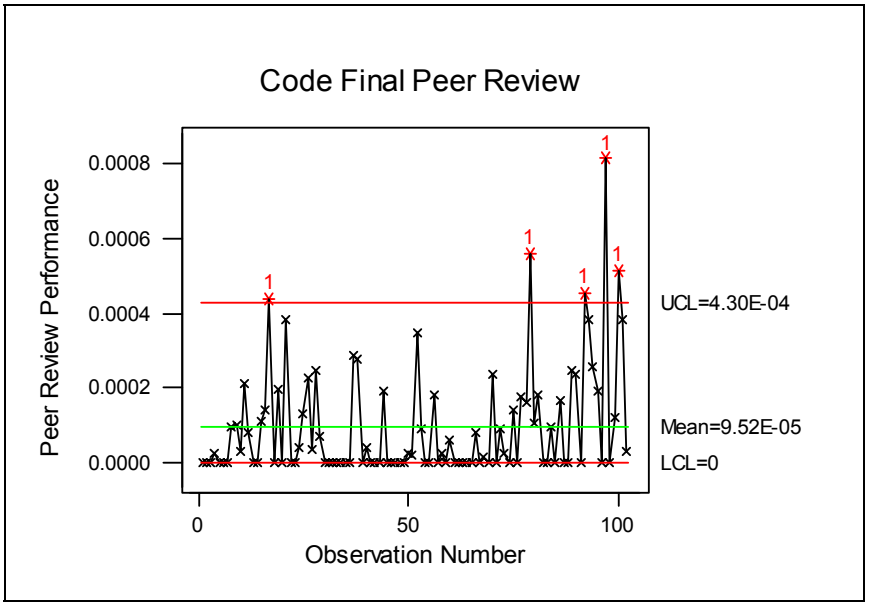

Figure 4.37 Code Final Peer Review Individuals Chart

All these findings gave us different opportunities to discover deficiencies and to implement improvement actions about review processes.

4.3. Analysis Results

While performing our case study, we faced a number of obstacles. First of all, performing a pioneering analysis required organization of historical data and it took significant amount of time. Most of the data was available online, but we had to go over hard-copy documents for some older projects. Moreover, the data collected was not completely appropriate for an SPC analysis. Therefore, we had to make some modifications, conversions and adjustments. We also needed to normalize measurement data and it necessitated extra effort to collect some additional measures. In some cases, these measures were not available and we had to restrict our analysis.

On the other hand, the number of collected data points was not sufficient to perform statistical analysis for some of the measures. The number of data further decreased when we removed them as outliers. We recognized two reasons: The

small number of projects, and the level of detail for collecting data. With high number of projects and more detailed data, the control charts would have been more reliable and explanatory.

Perhaps the most difficult part of the analysis was data analysis and charting. Working with too much data was a hard process and we spent hours on arranging and verifying thousands of numbers. We used some statistical tools to draw the charts and it simplified our effort to a certain extent.

The analysis became easier as soon as we established data tables and chart formulas. Finally we drew control charts and explored the process variation. We talked to related project individuals and examined relevant documentation to interpret the outliers. Our findings showed that control charts were quite effective to track process performance and to detect assignable causes. Because of the scarcity of data points and the high variability of data, however, we should be careful while interpreting the results. The current control limits may not have been representative of the actual process performance baseline and the charts may probably have missed some outliers due to wide control limits. With the addition of new data points in the future, more beneficial outcomes can be obtained. Despite the mentioned deficiencies, the analysis itself was very beneficial as we understood the meaning and benefits of measures, the means of applying SPC techniques and future improvement opportunities for better SPC analyses in a software company.

CHAPTER 5

SUGGESTED SPC GUIDELINES

5.1. General Comments and Suggestions

A CMM Level 3 company collects various metric data in order to visualize the processes, but without considering process control purposes. The following sections describe suggestions we derived as a result of our study so that software companies can gain maximum utility while trying to achieve successful SPC implementation.

To begin with, we have to specify that analyzing software data by a statistical method such as SPC is much more than drawing control charts. The analysis includes determination and definition of metrics, normalization of metric data, organizing data to produce control charts, deciding on the correct statistical approaches, then drawing control charts and finally interpreting chart outcomes. These activities necessitate both technical knowledge and sufficient time. For this reason, it will be appropriate to assign special individuals whose main responsibility will be statistical data analysis. Moreover, the initial preparation studies take quite a lot of time and require a considerable effort. Thus, the management should be patient for the outcomes and be encouraging under time and effort constraints.

Another important point for the reliability of measurements is related to the interpretation of the results and their reflection on the company personnel. The analysis results should not be to evaluate individual performances and a common

sense should be created on this idea so that people feel comfortable and confident while working.

Our analysis demonstrated that different types of projects have different characteristics. This distinction is obvious among internal and external projects. Therefore, better outcomes will be obtained if the projects are grouped with respect to certain characteristics and if separate analyses are performed for each distinct group.

Most of the effort for the analysis is spent during collecting metric data. An automated data collection mechanism would be very helpful to minimize this effort. In this way, SPC analysis may be feasible for more metrics. This will also provide data validity as manual collection is apt to produce more errors. On the other hand, a statistical tool will make it easier to draw control charts. Although spreadsheet programs may be sufficient in the beginning, a tool will be required when additional metrics will be tracked by SPC.

While analyzing the data, application of different techniques (i.e. different control charts, different size measures etc.) as complements will provide more dimensions and further understanding on the process. Their comparison also will be helpful to determine which gives better and more reliable results.

Frequently the number of data points may not be sufficient to calculate reliable control limits and it may not be possible to draw control charts. In such cases, other SPC charts such as Pareto diagrams or histograms may be utilized in order to compare process performances among different projects, but without control limits.

Despite the mentioned general guidelines, each metric possesses individual characteristics. For this reason, each metric should be treated separately. In the following sections we outline a step-by-step approach for applying SPC to the metrics that we analyzed during the case study. Nevertheless, the procedures should be modified for each organization considering its own dynamics and processes.

5.2. Rework Effort

Measurement

- Make a precise definition of rework.
- Distinguish between an enhancement and a defect; do not regard enhancement as part of rework.
- While recording defect data, clarify both the current project phase and the phase that the defect is related to.
- On the trouble report, define the origin of the defect (cause code).
- Record defects with different origins on different trouble reports.
- If possible, define each activity as rework or not while entering timesheet data and gather rework effort from timesheet directly. If this is not feasible, record the total problem resolution time (including correction and verification times) and obtain rework effort from trouble reports.

Preliminary Issues

- Perform separate analyses for different process types as much as the data allows (i.e. design document rework represents design process, code rework represents coding rework etc.). Determine the level of detail for generating document types by considering the organizational goals, existing problem domains and data availability.
- Provide a detailed timesheet mechanism so that effort amounts for different projects, CSCIs and builds (in evolutionary designs) can be obtained separately.

SPC Analysis

- Compute weekly rework and total effort amounts for each document type and code within each project, build and CSCI.
- If the rework effort is gathered from trouble reports, divide the rework effort among the weeks giving equal weights to each work day for the trouble reports that stay open for more than one week.

- Find the monthly organizational rework percentage for each document type/ code and obtain control limits for the Individuals charts.
- Find the monthly rework percentage values for each analysis unit (project, CSCI or build) and draw Individuals charts by using the organizational control limits.
- Perform a similar analysis in order to track rework percentage in each project phase separately (i.e. find out what percentage of total effort is devoted to rework in the implementation phase for each project, CSCI or build). Calculate control limits for each phase individually.
- Repeat phase based analysis regarding each project phase which rework is **related to (**i.e. find out what percentage of total effort is devoted to rework related to requirements analysis activities in the implementation phase for each project, CSCI or build). Calculate control limits for each project phase – related project phase combination separately.
- Analyze control charts and investigate causes of out-of-control incidents. Take relevant preventive actions for potential problems in the following phases and improve the process to avoid repetition of the same causes.

5.3. Defect Density

Measurement

- Make a precise definition of defect.
- For each defect, record the subject work product, origination date, closure date, defect type, priority, severity, origination point (cause code), project phase that the defect is discovered, project phase that the defect is related to, impact etc. (look up IEEE Guide to Classification for Software Anomalies [31] as a reference for defect categorization).
- Make a precise definition of size for different work products. Determine this measure for each document and code by considering the company specific conditions.

- For measuring software size:
    - Count blank lines, comment lines, reused code, and tool-generated code separately in CSU level.
    - If cyclometic complexity is measured, normalize SLOC count by giving a weight to cyclometic complexity for tracking defect density of logical type errors. For syntax type errors, do not perform this normalization.
    - If sufficient data is available, perform separate analyses for code components having different programming languages. Otherwise, convert SLOC to assembly language equivalent for the defects that are related to the logic. Do not perform this conversion for the defect types that are related to the actual code size (such as syntax).
- For measuring document size:
    - Consider different sizes for different documents. Number of requirements may be a good size measure for requirements documents, whereas a weighted number of words may work better for plan documents. If possible, try different size measures and decide which measure to use by analyzing the results.

Preliminary Issues

- Decide on the products for which SPC analysis will be critical and beneficial.
- The distribution of data in different defect categories may not be the same. Investigate this by dividing data into the defined categories. Decide on the level of analysis by considering the variation in distributions, the criticality of a specific category and the feasibility in terms of cost. In order to provide a balance between the cost of detail and the proposed benefits, form category groups (i.e. analyze priority 1 and 2 defects; impact 4 and 5 defects together).
- If possible, prepare separate documentation for different builds so that they can be individually analyzed.

SPC Analysis

- For each project phase, compute the cumulative number of defects for each product type – defect categorization combination (i.e. calculate the total number of priority 1 defects (defect categorization) for each requirements document (product type) found during design phase (project phase)).
- Measure the size of each product at the end of each project phase.
- List the products in time order and construct a data table like Table 5.1.
- Calculate defect density by the formula:

  defect density = number of defects found / size.
- Draw u-charts and Individuals Charts for each table.
- Perform the same analysis by using function points as size measure.

Table 5.1 Sample Defect Density Table (Priority 1 - Requirements Documents-end of design phase)

Document	Total number of defects	Size	Defect Density
SRS 1	15	47	0.319
SRS 2	23	83	0.277
…	…	…	…
…	…	…	…

5.4. Productivity

Measurement

- Make a precise definition of size for each product that is critical for analysis (refer to section 5.3).

- At the end of each project phase, measure function points separately for different CSCIs and project builds.
- Measure process efforts in sufficient detail so that CSCI and build based effort values can be gathered for each project phase.

SPC Analysis

- For each selected work product, calculate size at the end of the related project phase (SRS document size at the end of requirements analysis phase).
- Compute the effort spent during each project phase.
- Calculate productivity values for the selected products by dividing the size by phase effort.
- For each project phase – product combination (i.e. SRS document, requirements analysis phase), list the productivity values in time order.
- Make a regression analysis to find a correlation between the number of personnel and the productivity values. If sufficient evidence is obtained for a correlation, normalize productivity data by multiplying with a weight depending on the number of personnel.
- Calculate control limits and draw Individuals Charts.
- Add each new data to the end of the list and analyze productivity for the related product.

5.5. Review Performance

Measurement

- Make a precise definition of defect.
- Record sufficient detail for the defects (see section 5.3).
- Record the following information for each review at a minimum:
    - review type
    - reviewers

- review effort (including assessment and meeting times)
- number of defects found by each reviewer and the type of defects
- reviewed artifact
- review date
- Related trouble report

Preliminary Issues

- Categorize defects into different groups ([31])
- Categorize reviews considering their points in the life cycle and purposes (such as draft reviews, final reviews, change reviews, joint reviews).
- If possible, analyze each product type separately. Otherwise, categorize products into different groups (i.e. design documents, requirements documents, plan documents, code etc.) and analyze each group separately.

SPC Analysis

- Calculate the number of defects found per review time (including assessment and meeting effort in terms of man-hours or minutes).
- For each review type, categorize review performance data for the selected product types.
- Sort data in time order.
- Calculate organizational control limits for the Individuals charts for each review - product type combination.
- Divide metric data for each project in order to observe project specific performance.
- Draw Individuals charts and investigate causes of out-of-control incidents. Take relevant preventive actions for potential problems in the following reviews and improve the process to avoid repetition of the same causes.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Conclusion

The application of SPC to software processes has been a challenging issue for software engineers and researchers. Although SPC is suggested for providing process control and achieving higher process maturity levels, there are very few resources that describe success stories, implementation details, and implemented guidelines for applying SPC to specific metrics. With this realization, we performed a case study in a CMM Level 3 software organization in order to investigate the applicability of SPC within software processes.

In the case study we tried to find out the difficulties of using the existing metric data to implement SPC in a software company and means of utilizing SPC techniques for some specific metrics. We worked on defect density, rework percentage, review effectiveness and productivity measures. The results of this study demonstrated that each metric has particular characteristics and complexities regarding its definition, collection and interpretation. The difficulty increases further when we consider performing SPC analysis using the metric data. Most of these difficulties arise due to unsuitability of metric data for detailed statistical analysis. Actually the processes are defined and various metrics are collected in a CMM Level 3 organization. Nevertheless, there is little awareness about the specific attributes of metrics that are vital for detailed comparisons and statistical analyses.

Our study also contributed a practical perspective to the common debate on whether it is necessary to have a high maturity level for successful SPC implementation ([13], [17]). Radice [17] argues that it is possible to perform SPC in CMM level 1. He bases his claim on the idea that if a process is defined and performs consistently, SPC outcomes will be meaningful. However, we realized during our case analysis that performing well in a process is not sufficient to apply SPC techniques successfully. Reliability and meaningfulness of data requires normalization, and several other measures are necessary for normalizing a metric data. Thus the associated processes should also be well defined and properly performed. Most of all, the data collection and measurement procedures should be very well established. These evidences indicate that performing a process well is necessary but not sufficient for applying SPC to that process. That is why high process maturity is desirable for successful SPC implementation. This finding also provides evidence for the strength of CMM approach that requires maturity in many process areas to achieve a maturity level as opposed to SPICE, which enables process based capability with its two dimensions.

The analysis results showed that sufficiency of data is very critical for the reliability of control limits. This is dependent on the level of measurement and the number of projects in an organization. Although industrial averages for different metrics are available, it is not possible to calculate control limits without the data itself. Even if the data is given, the process capability values of other organizations cannot construct a baseline for a specific organization as it depends on various factors such as procedures, metric definitions and analysis approaches. The measurement data hides some inherent characteristics about the processes which are unique for each software organization. For this reason, finding enough data turns out to be one of the major problems for small-sized organizations and limits the utilization of SPC techniques to a few metrics.

We also realized that it is not possible to construct a general logic that will be used as a standard guideline for implementing SPC to any software measure. Each metric has its own dynamics, inherent characteristics and normalization techniques. The data are collected in various time intervals, analyses are

performed in different points in the life cycle and different charts are used for depiction. Therefore, separate approaches should be developed for each metric to be analyzed.

Actually, the complexity of processes and the existence of too many parameters for the value of the metric data inflate variability and make it difficult to apply SPC to software production. If the analysis is performed for specific processes, it becomes costly to collect, normalize and analyze the data. Moreover, the data set may not be meaningful since a small sample size is used. On the other hand, if SPC is applied to generic process data, the data will be scarce, there will be too much variability and we will not be able to detect any outliers. Thus, level of detail should be determined very carefully based on the amount of historical data on hand, the criticality of the process, and the detail of measurement data.

Considering all these factors, we see that the analysis is not straightforward. Nevertheless, a control chart acts as an auditor by showing existing nonconformities about a process / product provided that the necessary preliminary actions are taken. Thus it gives opportunity to detect problems and improve software processes. It is also an effective tool as it provides a visual interface with a scientific foundation. For that reason, software companies are trying to use SPC techniques as they mature their processes. It is not easy but the effort can be justified by expected benefits. With more experience and trials, better outcomes can be obtained from Statistical Process Control.

6.2. Future Work

During this study, we had some limitations because of time and data constraints. Nevertheless, this study is a step to the understanding of statistical process control in software industry and a good opportunity is left for researchers who want to get into the issue in more detail. The following are some of the possible items for further research studies:

- The effect of different parameters (i.e. programming language, defect priority, SLOC definition etc. for defect density; peer review type, defect type, number

of people in the review for review performance) on the results of SPC analysis can be investigated by a stepwise study.

- The study can be performed for each specific metric by analyzing SPC implementation among different software organizations.
- The efficacy of different tests on identifying outliers can be investigated (beside control limits).
- The analysis can be performed in a low maturity organization to understand whether SPC can still produce beneficial outcomes.
- The existing study can be improved by trying different measures for product sizes.
- A study can be performed in order to understand the sensitivity of control limits with 3 sigma standard deviation. Alternative control limits may also be investigated for better performance.
- Some other metrics can be investigated for their applicability to SPC analysis.

REFERENCES

[1] Sutherland, J., Devor, R., Chang, T., *Statistical Quality Design and Control.* Prentice Hall Publishing Company, 1992. ISBN: 002329180X.

[2] Card, D., "Statistical Process Control for Software?". IEEE Software, May 1994, PP 95-97.

[3] Kan, S. H., *Metrics and Models in Software Quality Engineering*. Addison-Wesley Publishing Company, 1995. ISBN 0-201-63339-6.

[4] Lantzy, M.A., "Application of Statistical Process Control to Software Processes", WADAS '92. Proceedings of the Ninth Washington Ada Symposium on Empowering Software Users and Developers, 1992, pp. 113-123.

[5] Crosby, P.B., *Quality is Free: The Art of Making Quality Certain*, Penguin Book USA Inc, January 1980. ISBN: 0-451-62585-4.

[6] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. *Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-024, ADA 263403).* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[7] CMMI Product Team, *CMMI^{SM} for Systems Engineering, Software Engineering, and Integrated Product and Process Development (CMMI-SE/SW/IPPD, V1.1), Continuous Representation.* Carnegie Mellon University, December 2001.

[8] Information Technology – Software Process Assessment – Part 4: Guide to Performing Assessments (ISO/IEC 15504-4:1998(E)).

[9] ISO/IEC TR 15504-5 Information Technology – Software Process Assessment, 1998.

[10] Burr, A., Owen, M., *Statistical Methods for Software Quality*. Thomson Publishing Company, 1996. ISBN 1-85032-171-X.

[11] Carleton, A., "Statistical Process Control for Software (Software Technology Review)". Carnegie Mellon University, 2001. (URL: http://www.sei.cmu.edu/ str/descriptions/spc_body.html).

[12] Florac, A.W., Carleton A.D., *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. Pearson Education, 1999. ISBN 0-201-60444-2.

[13] Florac, A.W., Carleton A.D., "Statistically Controlling the Software Process (The 99 SEI Software Engineering Symposium)", September 1999, Software Engineering Institute, Carnegie Mellon University.

[14] Florac, A.W., Carleton A.D., Barnard, J.R., "Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process". IEEE Software, July/August 2000, PP 97-106.

[15] Florac A.W., Park E.R., Carleton A.D., *Practical Software Measurement: Measuring for Process Management and Improvement (CMU/SEI-97-HB-003)*. Software Engineering Institute, Carnegie Mellon University, April 1997.

[16] Jakolte, P., Saxena, A., "Optimum Control Limits for Employing Statistical Process Control in Software Process". IEEE Transactions on Software Engineering, Vol: 28, No: 12, December 2002, PP 1126-1134.

[17] Radice, R., "Statistical Process Control for Software Projects". 10th Software Engineering Process Group Conference. Chicago, Illinois. March 1998.

[18] Romine, J., "Using Statistical Techniques to Manage Software Projects with Data", SEPG 2002 Conference, February 18-21, 2002.

[19] Weller, E., "Practical Applications of Statistical Process Control". IEEE Software, May/June 2000, PP 48-55.

[20] Humphrey, W., *Managing the Software Process*. Reading, Mass.: Addison-Wesley Publishing Company, 1989. ISBN 0-201-18095-2.

[21] Paulk, M. C., Chrissis, M. B., "The 2001 High Maturity Workshop, (CMU/SEI 2001-SR-014)". Carnegie Mellon University, January 2002.

[22] Ishikawa, K., *Guide to Quality Control*. Asian Productivity Organization, 1982. ISBN 92-833-1035-7.

[23] Montgomery, D.C., *Introduction to Statistical Quality Control*. John Wiley & Sons, Inc., Republic of Singapore, 1991. ISBN 0-471-51988-X.

[24] Shewhart, W.A., *Statistical Method: From the Viewpoint of Quality Control*, Lancaster Press Inc., 1939.

[25] Deming, W.E., *Out of the Crisis*, First MIT Press, 2000. ISBN: 0-262-54115-7.

[26] Fenton, N.E., Pfleeger, S.L., *Software Metrics*. PWS Publishing Company, 1997. ISBN 0-534-95425-1.

[27] Pajerski, R., Sova D., *Software Measurement Guidebook*, *NASA GB-001-94.* Software Engineering Program, August 1995.

[28] A Guide to the Project Management Body of Knowledge. Project Management Institute, Inc. 2000. ISBN 1-880410-23-0.

[29] FCD 9000-3: Application of ISO 9001:2000 to software (ISO/IEC JTC 1/SC 7 N2638 FCD 9000-3), 06-17-2002.

[30] IEEE Guide to Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, Std 982.2-1998.

[31] IEEE Guide to Classification for Software Anomalies, Std 1044.1-1995.

[32] IEEE Standard Classification for Software Anomalies, Std 1044-1993.

[33] Fenton, N.E., Neil M., "Software Metrics: successes, failures and new directions". The Journal of Systems and Software, 47, 1999, PP 149-157.

[34] Jones, C. *Software Assessments, Benchmarks, and Best Practices.* Addison-Wesley Longman Inc. 2000. ISBN 0-201-48542-7.

[35] Houston, D., “Cost of Software Quality: Justifying Software Process Improvement to Managers”. Software Quality Professional, Vol. 1, Issue 4, September 1999, PP: 8-16.

[36] Krasner, H., “Using the Cost of Quality Approach for Software”. The Journal of Defense and Software Engineering, November 1998.

[37] Barnard, J., “Managing Code Inspection Information”. IEEE Software, March 1994, PP 59-69.

[38] Beeson, D., “Achieving World Class Product Quality Using Software Inspections”. SEI SEPG 2001 Conference.

APPENDICES

APPENDIX A

STATEMENT OF WORK

1. Description and Scope

Summary of Work Requested

As part of his thesis, Umut Sargut will represent the findings of SPC implementation in XXX Software Company for the selected metrics. This will include the following items:

- CMMI suggested metrics will be taken as examples; however, currently collected metric data will be utilized during the analysis
- Relevant data to be collected will be described for each metric
- The represented features of the process/product and the relevance of metrics to the quality perspectives will be explained;
- The normalization methods for each metric and the related parameters required for normalization will be described;
- The conditions necessary for analysis results to be meaningful will be depicted;
- Statistical techniques applicable for the selected metrics will be described;
- The analysis results will be interpreted.

In order to perform the study, relevant metric data will be provided by the software company. Considering the confidentiality of the utilized data, the name of the company will not be mentioned in any part of the study. However, the actuality of the data from a software company will be pointed for its validity. The

metrics will be determined based on agreement. Although the actual data will be used, the scale of the data may be modified for confidentiality purposes.

Description of Major Elements (Deliverables) of the Completed Work

1. Proposal
2. SOW
3. A report (Report 1) including the descriptions about
   a. metrics;
   b. data to be collected;
   c. the relevance of metrics to processes and/or products;
   d. metric normalization methods and related parameters;
   e. prerequisite conditions for meaningful analysis;
   f. descriptions about statistical techniques applicable for the selected metrics
4. Interpretation of analysis results (Report 2).

Expected Benefits

XXX Software Company will have documented quantitative process control procedures for the selected metrics. Umut Sargut will gather related software metric data for the thesis.

2. Approach

Major Milestones/Key Events Anticipated

Table A.1 Milestones

Date	Milestone/Event
15.02.2003	• Report 1
15.04.2003	• Report 1 (Final); • Report 2 (First Draft – Including the SPC results for defect density and review performance metrics);
01.05.2003	• Report 2 (Final)

As the analysis is performed with the collected data, metric normalization methods and description details about statistical techniques are updated

accordingly. Therefore, Report 1 will have two versions in order to reflect the final updates on time.

Methodologies or Special Standards to be Observed

Standards:

Capability Maturity Model (CMM Version 1.1)

Capability Maturity Model Integration (CMMI Version 1.1)

Methodology:

The references suggested by Umut Sargut will be used for metric definitions.

During the study, some meetings will be performed for assessing the appropriateness of the work and for discussing different ideas.

Impact on Existing Projects or Systems

Some part of the data will be collected from an ongoing project. The previously collected data will be used as historical data in order to define control limits. The interpreted results may be input for project execution and may facilitate organizational level process improvement activities.

Plans for Periodic Status Reporting

Will be performed by telephone calls.

Procedures for Change of Scope and/or Work Effort (PCRs)

The metrics that will be subject to this study will be determined by agreement at the beginning of the project. Any scope change will also be determined upon agreement (in case of time constraints).

3. Resource Requirements

Detailed Plan for Human Resources Assignments

Table A.2 Allocated Resources

Person	Role
Umut Sargut	Responsible for the execution of the study.
Personnel 1	XXX Company representative. Consultant on software quality metrics. Responsible for providing relevant metric data.
Personnel 2	XXX Company representative. Consultant on project performance metrics. Responsible for providing relevant metric data.

Stakeholders

The stakeholders of the project are XXX Company and Umut Sargut.

4. Risks and Concerns

Because of timing constraints, all or some part of some the metrics may not be completed on time. In order to minimize the impact, the metrics will be given priority so that high priority metrics will be handled at first.

5. Acceptance Criteria

XXX Company will gather all mentioned deliverables at the end of the study.

6. Estimated Time and Cost

The study is estimated to be finished by 01.05.2003.

APPENDIX B

MEETING SCHEDULE

Table B.1 Meeting Schedule

Date	Subject	Duration (hours)
1/10/2003	Overview of SOW; discussion and determination of candidate metrics.	2
1/20/2003	Overall metrics analysis	2.5
1/22/2003	Overall metrics analysis	1.5
1/28/2003	Defect Density Discussion	2
1/29/2003	Rework Percentage Discussion	2
1/31/2003	Productivity Discussion	3
2/4/2003	Inspection/Test Effectiveness Discussion	3
2/7/2003	Data Analysis	2
2/18/2003	Data Analysis	1
3/6/2003	Data Analysis	0.5
3/11/2003	Data Analysis	1
3/19/2003	Data Analysis	2
3/27/2003	Data Analysis	3.5
3/28/2003	Data Analysis	2
4/1/2003	Data Analysis	2
4/2/2003	Data Analysis	2.5
4/3/2003	Data Analysis; meeting	2
4/4/2003	Data Analysis; meeting	2.5
9/4/2003	Data Analysis; meeting	1.5
10/4/2003	Data Analysis; meeting	1.5
11/4/2003	Data Analysis; meeting	1.25