# Software Quality System For Lebanese Companies

**Badia Z. Fidaoui**

**B.S., Beirut University College, 1994**

**PROJECT**

**Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at the Lebanese American University August 1997**

[redacted]

**Dr. Nashat Mansour (Advisor)**
Professor of Computer Science
Lebanese American University



[redacted]

**Dr. Ramzi Haraty**
Professor of Computer Science
Lebanese American University

# Acknowledgments

Developing this work would have been impossible without the support of many great people to whom I would like to express my sincere appreciation.

First, I would like to thank my family for supporting me all the study period. I would like to thank my father for encouraging me to start my higher education, my mother for taking good care of me, and my two sisters for bearing with me.

I would like also to thank my best friend, Rashed Hashash for his patience and help, for supporting me and for his continuous encouragement. Without his support and help throughout these three years, I certainly couldn't have made it.

My special thanks to Dr. Nashat Mansour, my advisor, for his guidance and advice, and to Dr. Ramzi Haraty for accepting to be on my committee, for all the ideas he has suggested, and for his support and assistance.

# Abstract

Based on a study conducted on the Lebanese software firms, it was found that the development process is applied based on subjective assessments and self made rules, documentation and planning issues are not the main concern, and assuring quality in software is regarded as a burden that delays the production of software and increases its cost. This showed the need of the Lebanese firms for a quality system to be introduced into their companies. The quality system is suggested for small-to-medium scale Lebanese firms in order to attain an acceptable level of quality in the software production industry. It is based on a selection of international standards namely the ISO and IEEE SQA standards. It contains and focuses on the minimum activities to be carried out in each of the development phases. The aim is to produce a simple and direct quality system integrated into the entire development phases and accepted by the Lebanese firms' managers. A case study is also presented, where the suggested quality system is implemented at a small scale Lebanese company.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1. Meaning of Quality

The software industry is one of the growing markets in the economy. Everyone is becoming more and more dependent on software systems. Development of high-quality and cost effective software has been a challenge facing the software industry. The question: 'is software facing a crisis?' is continually being asked. There is ample evidence to show that it is. Actually, such a question should not be asked, and more attention should be given to the problem and its solutions. Essentially, the problem stems from the quality of the software development process. Quality means conformance to requirements [Schulmeyer 1988]. The true quality of software is that which is perceived by the eventual users of that software: a system which is unreliable, for whatever reason, will be perceived as of poor quality, and such perceptions will be reinforced by a lack of responsiveness to problems identified by the user. Over the last decade, the software industry has witnessed a dramatic rise in the impact and effectiveness of Software Quality Assurance (SQA) [Smith 1990]. As a result, the successful development of software systems can be accomplished and users need not expect a product which is unmaintainable, unportable and unreliable. Thus, SQA has become an indispensable aid.

## 1.2. Problem Definition

The quality of industrial products has a wide economic consequences. Deviation by a small percentage from the prescribed quality factors can cause losses in the range of

millions in a competitive market. So, the use of software in business and industry is still unsatisfactory. Evidence for this is *the absence of a systematic development process supported by methods and tools* [Wallmüller 1994]. So, it is advisable to pay attention to software quality and to use appropriate principles, methods and tools. Software producers are expected to produce *high-quality products*, *on time* and *within budget*. We can see that the organization's problem in producing poor quality lies in the problems encountered in these three factors.

The problems in the *time factor* are that only five percent of all projects are completed on time, and more than sixty percent of all projects have at least a twenty percent time overrun. Many software projects are not completed on time, and completion problems often lead to the failure of the entire project [Wallmüller 1994].

The problems with the *cost factor* are that development cost increases exponentially with the complexity of software. The higher demand for adequate user friendliness and reliability also cause higher development costs. In many instances, sixty percent more of the entire software cost of a product is spent on maintenance [Wallmüller 1994]. Delays can reduce market opportunities for a product and render investment unprofitable. The problems of cost estimation are widely known. Attempts are being made to improve cost estimation by extrapolation of results from earlier projects and by estimating metrics such as line of code (LOC).

The problems with p*roduct quality factor* are that errors are found too late, frequently not until the customer tries to put the system into operation. The software product documentation is missing, incomplete, or not up-to-date. Because of product faults more than fifty percent of development time and effort is spent on error detection and correction. Finally, quality as a development aim cannot be proved because of lack of *Quality Planning* [Wallmüller 1994].

## 1.3. Causes

The causes of the present problems in software development are on several levels. The complexity of the software to be created is underestimated not only by management, but also by the developers of software products. The management of software projects does not meet the requirements. Software product requirements are not adequately specified. They are vague, confusing or contradictory. The developers often try to improve the quality by testing instead of developing the quality step by step. The documentation, one of the most important foundations for quality testing, is often missing or of poor quality.

## 1.4. Solution

The above mentioned factors lead to the realization that planning and production of software must be systematic and carried out by professional engineers and that quality must be a development goal. Thus, the goal of SQA is to help produce a quality software, on time and within budget.

## 1.5. Process and Product Quality

Different aspects of quality will be examined in order to achieve a distinct and comprehensive idea of software quality. In principle, there is a distinction between the *quality of a product* and the *quality of the development process.*

The quality goals for the software product determine the quality goals of the development process. The quality of the former is based on the quality of the development processes. These have a decisive influence on the quality of the product. Software development is a very complex process which requires the use of many different disciplines for the development of a product to satisfy its requirements. The

necessary disciplines are *Project Management*, *Quality Assurance*, *Configuration Management* and *Software Engineering* for the implementation process. Effective management and development practices are necessary.

## 1.6. Scope

The objective of quality management is to produce quality products by building quality into the products rather than testing quality into the products. Quality management is meant to ensure that faults do not occur in the first place. Quality management systems are used for developing products and are designed to ensure that quality is being designed and built into the products.

Based on a study conducted by Hajjar in Lebanon, some of the findings were that the development process in Lebanon is applied based on subjective assessments and self-made rules, details methodologies, documentation and planning issues are not the main concern, but rather the software as a whole. Add to this, that assuring quality in software is regarded as a burden that can delay the product from emerging and increasing its cost. Whenever, the software works, free of any errors and whenever the customer is satisfied, that is field proof that the software is of good quality.

This thesis provides a suggestion of a quality system based on international standards. The quality system is designed for a small-to-medium scale Lebanese company. It is designed to be effective for the company and meet the company quality objectives. The aim is to initiate such a system in Lebanon since in the Lebanese market there is no organized SQA. This will greatly improve the marketability of locally developed software.

## 1.7. Report Organization

This thesis is organized as follows: Chapter 2 offers a survey of related SQA concepts. Chapter 3 presents a study on ISO9001 standard and IEEE standard for SQA plans. Chapter 4 includes a suggestion of SQA program followed by a case study that implements the program in real situation in chapter 5. Chapter 6 is the conclusion. Some readings relevant to the case study are presented under the appendices. Appendix A, appendix B, and appendix C present respectively the Software Requirement Document, the Project Development Plan and the Software Design Document for the Callback billing system on which the suggested quality system was implemented. A list of Acronyms is presented in the Glossary.

# Chapter 2

# SQA Survey of Concepts

## 2.1. Introduction to the Concepts of Quality and Assurance

It is useful to define some concepts in connection with quality and assurance, examining classical definitions and the specific issue relating to software. It is important to recognize that quality is as important in the end product as it is during development. However, it is good to define some of the elements important to software quality, and to agree on the vocabulary used throughout this chapter [Wallmüller 1994].

A *principle* is defined as a concept on which the actions are based. They comprise general rules of attitudes, but they do not prescribe how a goal should be reached. For example, document structuring, and information hiding.

*Methods* support software engineering principles and lead the developers to predictable results. Methods rest on principles. For example, the method of structured analysis makes use of the principle of abstraction levels. A rough data flow is divided into more refined data flows.

*Tools* which support the application of principles, methods and are useful to the software developer, to the project leader and to the SQA engineers.

*Processing models* which structure the development and maintenance process through standardized processes.

*Standard* is an instruction about how a document should be laid out. For example, a requirement specification standard would specify all the sections expected in such a document and how each section is to be structured.

*Procedure* is a text which describes how a particular software task is to be carried out. For example, a procedure for programming would describe what standards to apply, where to store the source code and object code of a program or module, how to carry out certain categories of test and what documentation to fill in when the process of programming and testing have been completed. The important point about procedures and standards is that once they have been adopted for a particular project they have to be adhered to.

### 2.1.1. Quality

Difficulties have been encountered while explaining and defining *quality*. The difficulty of explaining quality is addressed through the following phrase:
"This is not because Quality is so mysterious but because Quality is so simple, immediate and direct" [Schulmeyer 1988].
Further difficulties are explained through the following dilemma exposed by Prisig:
"Quality is not objective..., Quality is not subjective..." [Schulmeyer 1988].
There is a range of formal and informal definitions available for quality. The Oxford English Dictionary states that quality is "the degree of excellence", but this definition does not take customer requirements into consideration. Customers generally require acceptable performance at a particular price point, rather than absolute excellence. An alternative formal definition of quality is provided by the International Standards Organization (ISO) as "*the totality of features and characteristics of a product or service that bear on its ability to satisfy specific needs*". The standard definition associates quality with the ability of the product to fulfill its function. This is achieved through the *features* and *characteristics* of the product.

In approaching the definition of quality from the point of view of J.M.Juran: "*The basic building block on which fitness for use is built is the quality characteristic*" [Schulmeyer 1988].

Actually, if we look at the definitions of quality given in many books, we find out that they all agree on the phrase "fitness for use". That is, a quality product is one which does what the customer expects it to do. Although fitness for use is an important concept, and forms a central point of view of Quality Assurance (QA), it is not the only property of a quality product. A high quality product is one which has associated with it a number of quality factors [Ince 1994]. Factors and Criteria or Characteristics and Features will be defined later.

### 2.1.2. Software Quality

Some definitions have been suggested which specifically refer to software quality. Examining some of these definitions, should aid comprehension.
In the IEEE Standard Glossary of Software Engineering Terminology, software quality is defined as: "Totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, conform to specifications".
Another sub-definition of IEEE for software quality: "The degree to which a customer or user perceives that software meets his/her composite expectations".
Turning more to the objective nature of software quality, an objective definition of software quality is "*the fitness for use of the total software product*". This definition recognizes the two features of a quality software, *conformance to its specification* and *fitness for its intended purpose*. These may be summarized as: Is it a good solution? and does it address the right problem?

### 2.1.3.Quality Assurance

Often 'Quality Assurance' is spoken of as if the two words were one: 'quality-assurance'. This refers to the entire process of *assuring* software *quality*. There are two elements involved here: quality and assurance.

Quality

Crosby defined quality as conformance to requirements. Crosby's definition of quality implies two areas of action for the SQA function: First, ensuring that the *requirements* established for the software correspond to user *needs*. Second, ensuring that the software *product* adheres to *requirements*, both general functional requirements and specific quality characteristics. The distinction made between the two types of requirements is that while a software product may perform its general processing function, there are a number of software characteristics which may also be required of it [Sinclair 1988].

The quality function is an iterative process, whose elements may be defined as follows: Document-Discuss-Agree.

To Document means to write down ideas about what requirements and characteristics should constitute software quality for a particular project. What is meant by Discuss is to talk about the specifications and characteristics with others involved in the project. As to Agree is to reach a consensus with the discussion group about the requirements and characteristics which will define quality, and modify the initial documentation accordingly.

For the general functional requirements of the software, discussions must be conducted and their outcome will be a hierarchy of formal system documentation detailing the performance specifications to be met.

For the specific quality characteristics required, the outcome of the discussions should be threefold: First, a series of 'characteristics' which must be present in sufficient degree to constitute acceptable or 'quality' software. These are software quality

Factors. Second a series of subcharacteristics which constitute the quality Factors, and whose presence to a sufficient degree will ensure that the Factors they constitute are present. These are Criteria. Third, a series of checklists based upon the Criteria. These allow the verification and measurement of the quality Factors and their Criteria.
In a later section, we will present some of the agreed-upon software quality Factors and their Criteria.

Assurance

As to Assurance, it is the process by which we achieve the desired goal of quality. Like the quality function, assurance is an iterative process. It is composed of four elements : Process-Documentation-Review-Comparison.
These elements may be defined as follows : Process is a series of actions carried out according to their definitions in the SQA development process. Documentation is a written record of the process. Review is an examination of the documentation produced. Comparison is a matching of the documentation produced with standards for conformance to determine acceptance. If the documentation conformance is found to be lacking, the initial assurance step, Process, becomes corrective and the loop is repeated.

Many definitions have been suggested which specifically refer to quality assurance. The definition of quality assurance within the context of IEEE STD729 can be summarized as follows: "Quality assurance is a planned and systematic pattern of all actions-procedures, techniques and tools necessary to provide adequate confidence that the item or product conforms to established technical requirements".
Reifer provides the following definition: "Software quality assurance is the system of methods and procedures used to assure that the software meets its requirements. The system involves planning, measuring and monitoring developmental activities performed by others" [Wallmüller 1994].

The narrowing of the software quality definition provides the definition for software quality assurance: "*The systematic activities providing evidence of the fitness for use of the total software product*" [Wallmüller 1994].

### 2.1.4. Quality Policy

What is meant by quality policy are the basic aims and objectives of an organization regarding quality, as stipulated by management. Examples are customer orientation, fast reactions to market conditions through the introduction of new products, and manufacturing of products with comprehensive and efficient customer service.
Quality policy is a central task of top management. Quality policy in connection with software means that the three project elements of time, cost and product requirements, are equally important.

### 2.1.5. Quality Assurance System

Quality Management System, Quality Assurance System (QAS), and Quality System are defined to be "the system" of people, processes, procedures, tools, disciplines, and practices that are directly involved in producing a product or providing a service. They are systems for managing quality (i.e., building quality into a product). This includes the organization of construction and release procedures, the allocation of responsibilities and the selection of tools for the implementation of quality assurance [Ince 1994].
Quality assurance systems provide the framework for all quality-assuring measures and strategies. It includes all of those steps necessary to begin a SQA effort in an organization, from initial inception (the decision to have an SQA program) to actual implementation (the point at which the actual QAS is fully in place and running).

### 2.1.6. Quality Assurance Plan

The Quality Assurance Plan is the central aid for planning and checking QA. It contains all deliberately chosen QA measures for a software project, and consequently it is the written proof of quality control. The QA Plan is a document outlining the details of all procedures, policies, reviews, and audits which will be used to track software projects through their life cycle, from initial request to actual operation, assuring that quality is built in. It forms part of the overall project plan for a software project.

### 2.1.7. Project Development Tasks of SQA

What are the main tasks of quality assurance? We can distinguish between quality planning, quality control and quality testing.

#### 2.1.7.1. Quality Planning

Quality planning is a process of assessing the requirements of the product, and the context in which they must be observed. For this, quality features must be selected, classified and weighted. Quantification of the features plays an important part, since it provides the evidence for compliance with the plan. Aids for measuring and evaluating, quality metrics must be provided. Quality planning is product and process dependent and must be agreed with the client or user.

The quality assurance planning includes the support in defining quality requirements, the support in the creation of the project-specific quality assurance plan and advice on the selection of constructive and analytical measures, the support when purchasing software and when subcontracting to other companies.

### 2.1.7.2. Quality Control

What is meant is the control, supervision, and correction of the implementation of work with the goal of meeting the given requirements. Important aids to supervision are quality tests. The results are used to recommend correction measures. Quality control is closely linked with project management.

The supervision and evaluation of correction measures task includes the collection and evaluation of shortcomings which were noted when assessing product quality, and the following-up of requests for alterations and correction measures.

Juran provides a definition of quality control: "It is the process through which we measure actual quality performance, compare it with standards, and act on the difference" [Schulmeyer 1988].

Reifer equates software quality control with software verification: "It is the set of verification activities which at any point in the software development sequence involves assessing whether the current products being produced are technically consistent and compliant with the specification of the previous phase" [Schulmeyer 1988].

Further narrowing leads to the following definition of software quality control: "*Independent evaluation to assure fitness for use of the total software product*" [Schulmeyer 1988].

### 2.1.7.3. Quality Testing

Quality testing is the assessment to the extent to which a test object meets the given requirements. We can distinguish between static and dynamic testing. Examples of static testing are reviews and audits. To the category of dynamic testing belong, tests and counts of test features through the use of tools.

The quality testing task includes the checking of phase results, analysis of applied processes, aids and tools, audit of projects, and quality assessment of software products.

### 2.1.8. SQA Objectives

The basic function of SQA, is to help the future user select those qualities important to him, to ensure that both the user and the provider share the same understanding of qualities, and to help the provider to ensure that those qualities are present in the final product to a sufficient degree to make that product acceptable.

Quality assurance should not be viewed only as an auditing function, with the aim of detecting errors or problems which have already occurred. The quality function should aim to prevent problems before they occur through education, and the introduction and support of appropriate procedures, standards, techniques and tools; to assure that processes are in compliance with established quality procedures and that products are being developed in accordance with approved product specifications and requirements.

### 2.1.9. Quality Models

The quality of a software product is the totality of characteristics and features which relate to the suitability of this product to meet the prescribed requirements. In order to test the specification and the target levels of quality requirements, aids are needed which are related on the one hand to the development process and on the other to the product. Quality models make specifications possible which take into account the requirements of the software product as well as the process. Different quality models have been developed, by Boehm, McCall, and others. So, they were introduced as major aids for quality planning and evaluation. A hierarchical model of software quality is based upon a set of quality Factors and their corresponding Criteria, each of which

has a set of measures or metrics associated with it. This type of model is illustrated schematically in Figure 2.1.

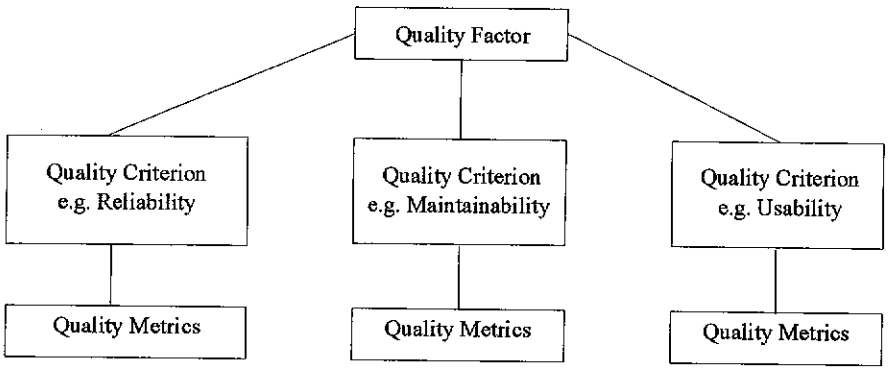


**Figure 2.1** A hierarchical model of quality

Since the main function of SQA is to ensure that the user is given a product that meets its specified needs, to ensure that the user selects the qualities important to him/her, to ensure that the provider and the user share the same understanding of qualities, and to help the provider to ensure that the qualities are present in the final product, it is important to talk about quality models. The reason for discussing quality models, is that they provide a way for reaching a consensus on the software quality Factors and their related Criteria which is a very important and preliminary step in ensuring against a problem software product.

So, an important aspect of quality models is the partitioning of quality characteristics which has metrics as its lowest level. Boehm was the first to suggest and show 21 characteristics and about 60 quality metrics in his model for evaluation of software [Gillies 1992].

Depending on the prescribed quality model and the specific product requirements, a specific number of characteristics, features, and metrics is determined for each project. In general, requirements regarding ease of maintenance, portability are mostly forgotten or ignored. Only functional requirements, performance requirements and

requirements of the user interface are specified. The reason is that the people responsible for the specification do not have any aids for formulating these quality requirements. Quality models and quality metrics provide the opportunity for specifying these requirements.

We have seen that the quality assurance function is an iterative process, and we have seen its elements whose outcome are quality Factors and Criteria. What are the generally accepted Factors and Criteria that constitute software quality?

### 2.1.9.1. Factors of Software Quality

Factors may be described as higher-level, more general elements. Criteria in turn are those characteristics which define quality Factors. Table 2-1 identifies, and offers definitions of, eleven agreed upon Factors of software quality. This table is taken from the study by McCall [Vincent 1988]. A listing of quality Factors include Correctness, Reliability, Efficiency, Integrity, Usability, Maintainability, Testability, Flexibility, Portability, Reusability, and Interoperability.

| Quality Factor | Definition |
|---|---|
| Correctness | Extent to which a program satisfies its specification and fulfills the user's mission objectives |
| Reliability | Extent to which a program can be expected to perform its intended function with required precision |
| Efficiency | The amount of computing resources and code required by a program to perform a function |
| Integrity | Extent to which access to software or data by unauthorized persons can be controlled |
| Usability | Effort required to learn, operate, prepare input, and interpret output of a program |
| Maintainability | Effort required to locate and fix an error in an operational program |
| Testability | Effort required to test a program to ensure it performs its intended function |
| Flexibility | Effort required to modify an operational program |
| Portability | Effort required to transfer a program from one hardware configuration and/or environment to another |
| Reusability | Extent to which a program can be used in other applications related to the packaging and scope of the functions that the program performs |
| Interoperability | Effort required to couple one system with another |

**Table 2-1** Factors of Software Quality

### 2.1.9.2. Criterion of Quality Factors

As we said above, Criteria are those characteristics which define the quality Factors. In other words, quality Factors may be divided into independent Criteria which may be easily measured. The total score of Criteria for a quality Factor will determine the extent to which that Factor is present, that is the extent of a software product's quality at that point. There are several reasons for developing a list of the Criteria for each Factor: First, Criteria offer a more complete, concrete definition of Factors. Second, Criteria common among Factors help to illustrate the interrelation between Factors. Third, Criteria allow audit and review metrics to be developed with greater ease.

Finally, Criteria allow to pinpoint that area of a quality Factor which may not be up to a predefined acceptable standard. Table 2-2 suggests a number of generally agreed upon Criteria for software quality Factors.

| Criterion | Definition | Related Factors |
|---|---|---|
| Traceability | Those attributes of the software that provide a thread from the requirements to the implementation with respect to the specific development and operational environment | Correctness |
| Completeness | Those attributes of the software that provide full implementation of the functions required | Correctness |
| Consistency | Those attributes of the software that provide uniform design and implementation techniques and notation | Correctness<br>Reliability<br>Maintainability |
| Accuracy | Those attributes of the software that provide the required precision in calculations and outputs | Reliability |
| Error Tolerance | Those attributes of the software that provide continuity of operation under nonnominal conditions | Reliability |
| Simplicity | Those attributes of the software that provide implementation of functions in the most understandable manner(avoid practices that increase complexity | Reliability<br>Maintainability<br>Testability |
| Modularity | Those attributes of the software that provide a structure of highly independent modules | Maintainability<br>Testability<br>Flexibility<br>Portability<br>Reusability<br>Interoperability |
| Generality | Those attributes of the software that provide breadth to the functions performed | Flexibility<br>Reusability |
| Expandability | Those attributes of the software that provide for expansion of data storage requirements or computational functions | Flexibility |
| Instrumentation | Those attributes of the software that provide for the measurement of usage or identification of errors | Testability |
| Self-Descriptiveness | Those attributes of the software that provide explanation of the implementation of a function | Maintainability<br>Testability<br>Flexibility<br>Portability<br>Reusability |
| Execution Efficiency | Those attributes of the software that provide for minimum processing | Efficiency |
| Storage Efficiency | Those attributes of the software that provide for minimum storage requirements during operation | Efficiency |

**Table 2-2** Criteria of Software Quality Factors

| Criterion | Definition | Related Factors |
|---|---|---|
| Access Control | Those attributes of the software that provide for control of the access of software and data | Integrity |
| Access Audit | Those attributes of the software that provide for an audit of the access of software and data | Integrity |
| Operability | Those attributes of the software that determine operation and procedures concerned with the operation of the software | Usability |
| Training | Those attributes of the software that provide transition from current operation or initial familiarization | Usability |
| Communicativeness | Those attributes of the software that provide useful inputs and outputs which can be assimilated | Usability |
| Software system Independence | Those attributes of the software that determine its dependency on the software environment (operating system, inputs/outputs, etc.) | Portability Reusability |
| Machine Independence | Those attributes of the software that determine its dependency on the hardware system | Portability Reusability |
| Communications Commonality | Those attributes of the software that provide the use of standard protocols and interface routines | Interoperability |
| Data Commonality | Those attributes of the software that provide the use of standard data representations | Interoperability |
| Conciseness | Those attributes of the software that provide the implementation of a function with a minimum amount of code | Maintainability |

**Table 2-2** (*Cont'd*) Criteria of Software Quality Factors

Figure 2.2 provides an illustration of the relationship between these Criteria and the Factors.

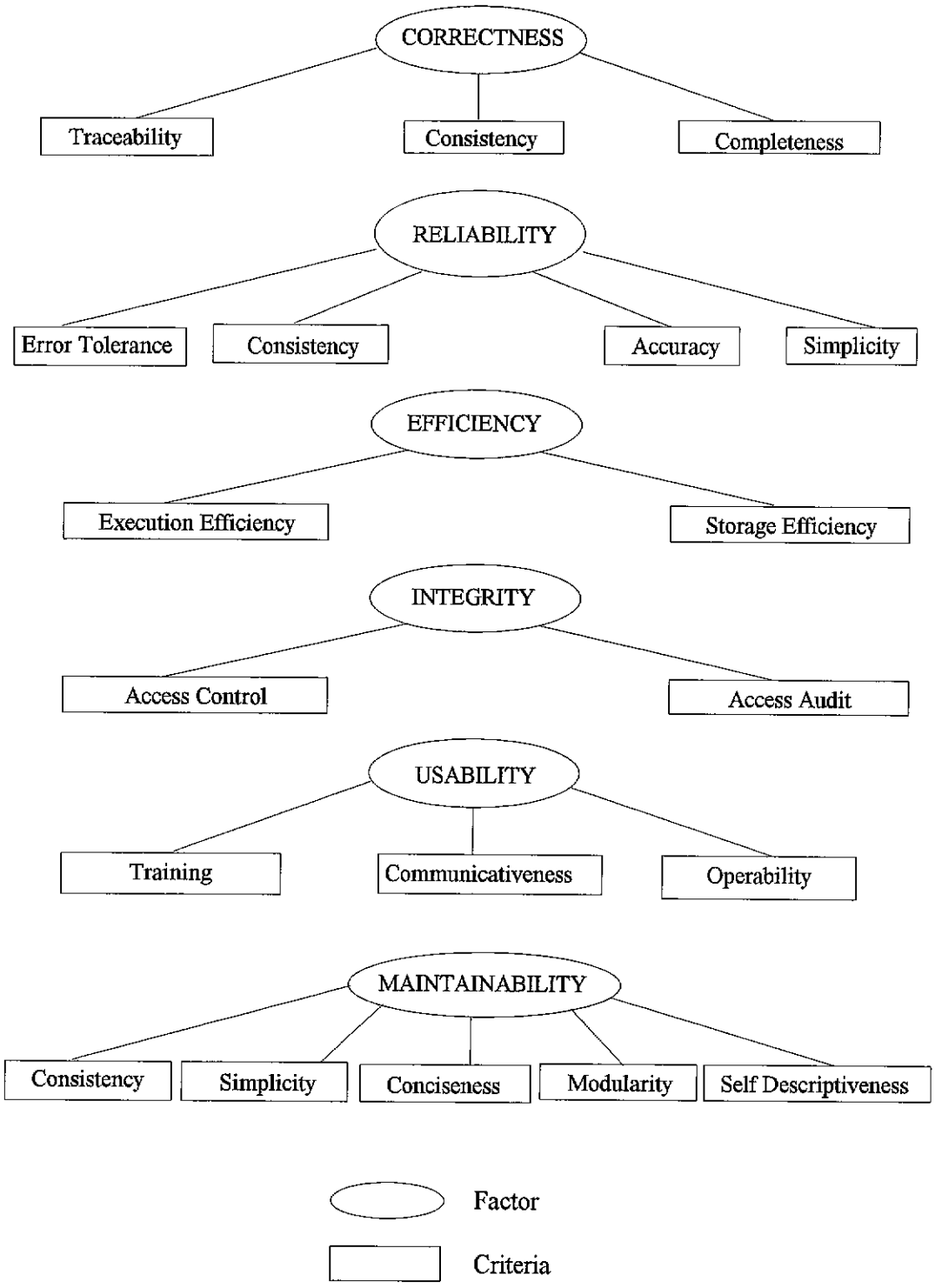


**Figure 2.2** Factors of Software Quality and their Related Criteria

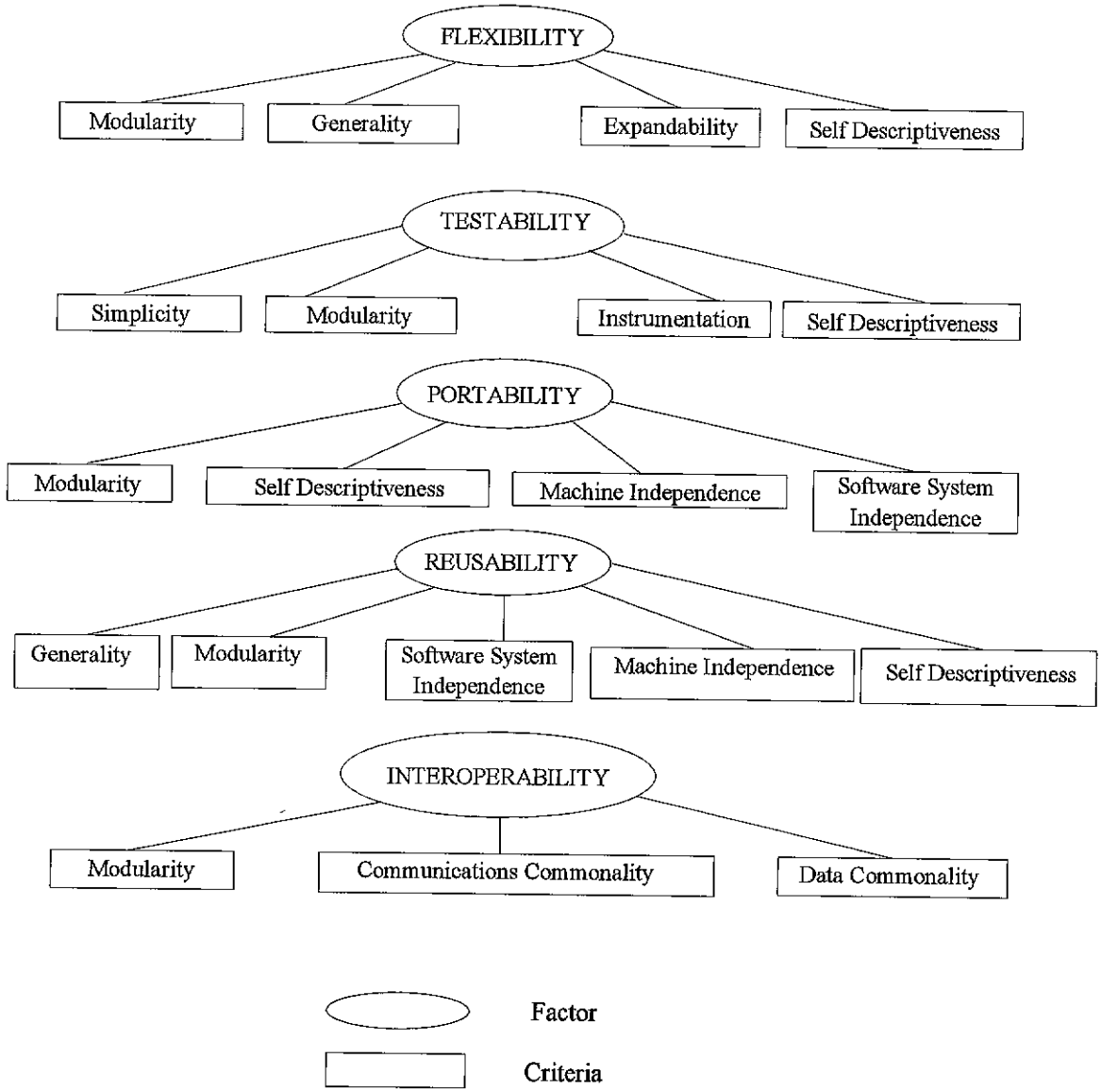


**Figure 2.2** *(Cont'd)* Factors of Software Quality and their Related Criteria

### 2.1.9.3. Advantages and Disadvantages of Quality Models

Through the application of quality models many objectives are attained. First, the different ideas of software quality are made consistent. Second, there is uniform communication. The user and the provider share the same understanding of quality. Third, quality becomes concrete, it is definable and can be planned. Quality goals can

be specific by making available meaningful, quantifiable quality Criteria for the product or process. Quality can be evaluated objectively. In this way, the development factor quality can be planned and controlled. Management can see graphically to what extent quality goals have been reached. The documentation of the product improves. Then, meeting user requirements becomes easier to check. Proof of the effect of QA measures becomes transparent and therefore simpler. Finally, quality models can also help us to a better understanding of the relationships between Factors.

There are trade-offs to be considered between quality Factors. In the case of a negative impact between Factors, the user will decide based on the importance of each affected Factor to the end use of the software product, what the nature of the trade-offs will be. If, for example, human lives are to be affected by the software product, a high degree of Testability is desired, a Factor whose presence will enhance the Correctness and Reliability of the product by allowing a thorough test to ensure that it will perform its intended function. However, in some cases, some of the Efficiency Factor must be sacrificed, because it conflicts with Testability.

A statement explaining the applications concerns and outlining the related quality Factors and their Criteria trade-offs should be drafted for each software developed under the SQA function [Wallmüller 1994].

Although quality models help to improve process quality. Typical process quality problems are unsatisfactory project planning and control, underestimation of development cost at the beginning of the project, and delayed delivery of the product. By choosing suitable metrics for specific process factors such as project planning or though standardized procedures within the project, process and product problems can be more easily handled.

Despite these advantages there are a number of problems which must be solved. One disadvantage is that the connection between metrics, Factors, and Criteria have not yet been proved theoretically, and thus are only at the stage of hypothesis. The development and validation of quality models is a current area of research in software

engineering [Wallmüller 1994]. The selection of meaningful quantifiable Criteria is different and diverse for different development processes and is in its beginning.
However, despite many unsolved problems, quality models are already a useful mechanism for improving the control of quality.

### 2.1.9.4. Views of Quality

One important aspect of quality planning and evaluation is a set of ideas on quality. As it is known, quality is not absolute and is always based on predetermined requirements. These predetermined requirements are dependent on the person who prescribes them. People from different environments dealing with the development of a software product have different ideas of quality. According to a study made by Dissman and Zurwehn when constructing quality models, they define four groups of people who see the situation from different views. These are the users, managers, designers and programmers [Wallmüller 1994].
For each of the various groups of people different quality views can be stated. Quality views comprise a large number of communicating quality Factors which are relevant for planning and evaluation of quality for the group.
The first group are the users of a software product. Quality requirements of this group usually affect the interfaces of the product.
The second category of people are managers of a software product. Managers are those who provide software products which are reliable and maintainable and will keep the customer satisfied. However they have deadlines and budget constraints to meet. The requirements of this group are application oriented and concerned with the future of the product.
The third group of people are the designers. They determine the technical structure of the product as the combination of components and the functions of the components. The quality goals of this group affect, the structuring of architecture, and aim at the

satisfaction of user and management requirements and control in the direction of development and further development of the product.

The fourth group of people are the programmers. They create the actual components of the system according to the prescribed functions in the form of programs and modules. Their quality goals usually concern the detailed programming structure, the programming style and the particular algorithms.

One can notice that these views may conflict with each other. While the customer is interested to know to what extent the quality and application of the product are affected by cost, time constraints, and technical risks, the project manager is interested in reaching agreements regarding cost and time limits, and also an acceptable quality level which is competitive with other user products.

The limits of quality models today lies partly in the inability to describe software through quantifiable Criteria, and partly in the poor structuring and formal description of development and maintenance processes. Many of the links between Factors, Criteria and metrics in quality models rest on values based on experience, most of them not systematically evaluated [Wallmüller 1994].

Summarizing, we can say that, by taking into account the different views, applying a quality model becomes easier and simpler.

### 2.1.9.5. Meaning of Quality Models in a QAS

Quality models are an actual research and development area of software engineering. The central issues of QA, such as quality planning, quality control and quality testing, can be solved only by metrics. Quality models offer a suitable foundation (Factors, measurable and assessable Criteria).

The most important quality Factors for a project must be chosen. Thereafter they are used for quantitative quality goals, assuming that the respective Criteria for these quality Factors exist and that they are measurable. The relevant QA measures can then

be chosen in order to make sure that quality goals are reached. These measures comprise the application of methods, tools and aids, and quality control lies in their suitable practical application.

At certain points in time, intermediate evaluations are carried out in the form of quality tests on phase deliverables, as it is planned in the QA Plan. They show us to what extent quality goals have been reached in each phase. At the same time, they cause the initiation of correction measures in order to reach the set quality goals. The aim is to come to an acceptable quality level by using the systematic processes and practical aids of software engineering [Gillies 1992].

### 2.1.10. Quality Assurance Principles

The creation of software is subject to the demand of high productivity. This involves the goals of timescales, minimum cost and satisfactory product quality. If these goals are met, we speak of economical software development. To achieve goals like high productivity and the meeting of budgets, timescales and product quality requirements, certain general principles of SQA must be observed. The following principles concerning assurance of quality are at present acknowledged [Wallmüller 1994].

*Practical quality Factors*

Of chief importance to quality is the observation of customer and user quality requirements from the beginning of project planning. It is important that quality Factors are concrete and if possible quantifiable.

*Product and project-dependent quality planning*

Insufficient attention is given to examining the real application process of the software product to be planned and developed, such as the lifetime of the product and who the potential user will be. All these factors influence the quality requirements of a project. Depending on these quality requirements, the appropriate software engineering

methods and tools can be applied and suitable QA measures taken. For example, in a bank, the quality characteristics for a customer information system would be easy handling and reliability.

In quality planning, if project uncertainties are not taken into account, quality assurance plans are useless. The risk factors which influence product and quality assurance planning include: the number of staff, the qualifications of staff, degree of uncertainty of requirements, and the economic risk such as the time constraint for product release on to the market. These risk factors are different in each project. In the case of a risk, the respective QA measures must be chosen and project planning must be modified or made more precise.

*Checking of results of quality tests*

It is an important part of successful quality control. Discrepancies from the planned quality level will be revealed. The aim is to arrive at suitable correction methods in the development process based on the findings in quality reviews. Reviews and audits provide the necessary information for correction of the development process.

*Multiple quality reviews*

Here, we are dealing with informal tests. The capabilities of individuals are better utilized. This occurs, for example, when the developer of an intermediate or end deliverable of a phase asks his colleague to look at the documents and proofread them.

*Maximum constructive quality assurance*

While quality testing involves determining the existence or non-existence of quality, the aim of constructive QA is to avoid errors in the development process. By the use of suitable preventive measures, the product quality will be directly improved and the quality tests will be reduced.

*Early discovery and correction of errors and faults*

An error in the requirement phase which is not discovered until the product is put into operation by the end user costs about one hundred times more than early discovery in the requirement phase. Therefore, the strategy must be to recognize and eliminate errors as early as possible.

*Integrated quality assurance*

Quality assurance must be integrated into the entire development process. This leads to QA measures being planned and organized with other development measures. The basic principle is that each development activity consists of a constructive and a testing part. Both parts are reflected in the respective documentation. For example, requirements are listed by means of a suitable requirement definition method. By means of a formal or informal requirement review, faults in the requirements are subsequently removed. This principle makes the level of quality visible at each point in time, and is important for the earlier phases. It is a prerequisite for checking whether quality goals have been reached.

*Independent quality testing*

The central issue in quality testing is the discovery of errors and to show the current actual quality and to deal with error elimination at a later stage. If developers have to test their own work, it often leads to tension and psychological problems. This conflict can be resolved through independent quality testing. By applying this principle, the consequences of lack of objectivity can be avoided. One disadvantage can occur if it is thought that it is the person who is being judged rather than the product.

*Evaluation of applied quality assurance measures*

At certain time intervals, it is necessary to check the applied quality assurance organization and its measures. Audits result in corrections of the quality system and the applied QA measures.

It is important that management understands the quality principles mentioned above, and takes quality seriously. When this happens, quality plan and program are likely to succeed, and producing quality products on time and within budget is possible.

### 2.1.11. Quality Assurance Measures

The quality assurance measures forms the basis for the SQA team. Four categories of QA measures can be distinguished: planning and administrative, constructive, analytical, and psychology orientated.

Planning and administrative QA measures, where we are dealing with the construction, introduction and maintenance of a quality assurance system.

Constructive QA measures are those which serve the creation of quality. They are preventive and should avoid the creation of errors and quality faults from the start by stipulating suitable principles (i.e. structuring in a document), methods (i.e. Jackson system development), tools (testing tools, configuration management systems for administration, pert chart for planning, CASE tools and QA tools), techniques (i.e. prototyping), and process models (i.e. spiral and waterfall). They also include all measures for error correction.

Analytical measures lead to the improvement of a product. It includes Verification and Validation, Static tests which focus on reviews and audits, Dynamic tests that define the testing methodology/activities and results; for each test (module, integration, system, and acceptance), test plan, test design, test case specification, test procedure specification, test execution, test analysis must be specific. The testing methods might be black box or white box.

The category of Psychological QA measures concerns the person as a developer, project leader, or project manager. We can distinguish between measures which concern the work of the individual and those which concern team work.

### 2.1.12. Necessary Plans

The quality of a software product is determined by its development process and the Factors it possesses. The development process can be structured in time by the introduction of phases. Quality must be achieved in each of these phases. It is important that there are requirements and that these are checked for their completeness at the end of the phase. Certainly, the achieving of milestones is an important process requirement for management. For example, the creation of a document is a milestone at the end of a phase.

So, the development process, in particular its quality, plays an important part in the creation of product quality. Prerequisites for good process quality are the extent of systematic, methodical development, that is development standards, careful project management, qualifications and training of staff, and the quality of aids used.

How and when errors are created in the process? The creation of faults and errors is due to an accumulation effect. The project normally begins with collection, analysis and definition of requirements. These requirements are recorded in the form of a requirements specification. Experience has shown that part of the specification is correct, while the other part contains errors.

In the next phase, the design is created. The result is a design document. Part of the design is correct, but another part contains errors which were created during this phase, and another part of the design is based on faulty specifications.

The next step is programming. Part of the program is correct, but another part contains errors. Another part of the program is based on faulty design, faulty specifications or faulty requirements.

In the following integration and test phase, part of the program which functions correctly, another part contains errors and can be corrected and which are being corrected. Another part contains errors that cannot be corrected. Yet another part contains hidden errors. Overall, the software product is not perfect. This imperfection

is due to the cumulative effect of errors and faults and the overall effect can only be observed at the end of the project.

The cumulative effect of errors can be reduced by suitable QA principles. One of the most important principle is to apply tests at the beginning of the life cycle as well as to test the process results. If this is not done, the correction of quality faults can become very expensive, since the cost of correcting errors increases exponentially with the time that an error remains in the product.

A major source of faults and errors lies in planning: that is in the planning of the project as a whole and in the development process in particular. An examination of problems in management of software projects by Thayer *et al.* found that sixty percent of problems are due to project planning and twenty percent to project control. From this, we can see how important planning is for the achievement of a successful project [Wallmüller, 1994]

We start the software development process with specific requirements, such as the requirement of a given product quality, or the requirement to complete the project by a given time within a certain budget. How we can carry out the development process in a way which allow us to meet these requirements is our aim. We are dealing with the planning component in project management, and the interrelationship of project management, quality assurance and configuration management.

One of the most important aids for quality-oriented project planning is the *Project Plan.* The important aspects of this plan are: project organization (the setting up of an organization chart, determining responsibilities); project description (testing process, requirement of project environment); development process in the form of a development plan (division of development work into manageable parts, deadlines, budget, and a list of risk factors); quality assurance in the form of a quality assurance plan; CM with a Configuration Management Plan (CMP); and project-specific training [Wallmüller 1994].

So, Project management can be a source of problems and errors in connection with QA and CM, and QA is a management issue.

### 2.1.13. Degree of SQA involvement

The degree to which a SQA team gets involved in a project is not fixed. An SQA team may remain completely independent from the development team in terms of tasks, or may take a more active role in performing some of the developmental activities. Such decisions must be tempered in the best interests of software quality management and software development management as a function of the project.

So, it is important that there should be managerial independence between the development team and the SQA group. That is to say, development should be under one manager and SQA under a different manager, and neither manager should be able to overrule the other.

The important thing is that SQA must never loose sight of its role as an independent management team when it comes to matters of quality. No matter how involved it is in the project itself, the sole justification for a quality team on a project is to assure that processes are in compliance with established quality procedures and that products are being developed in accordance with approved product specifications and requirements.

SQA plays a lead role in each phase. Before the next phase is entered, each phase must be required to pass some form of test or inspection.

The implication is that SQA management must concern itself with how well the specific objectives of each phase are being accomplished on a day-to-day basis. Such concerns should be: what are the objectives of this phase? Are the objectives well understood? If the objectives are unclear, from whom the resolution must be seeked?

The objectives must be stated and approved by the entire management team, and coordinated among the project plans: software development plan, test plan, CMP, SQA plan.

# Chapter 3

# Standards

## 3.1. Managing Quality

The objective of quality management is to produce quality products by building quality into the products rather than testing quality into the products. Quality management is meant to ensure that faults do not occur in the first place. Quality management systems are used for developing products and are designed to ensure that quality is being designed and built into the products.

Quality management system, quality assurance system, and quality system are defined to be "the system" of people, processes, procedures, tools, disciplines, and practices that are directly involved in producing a product or providing a service. They are systems for managing quality, i.e., building quality into a product [Ince 1994].

International standards such as ISO 9001 and IEEE standard for Software Quality Assurance Plans (SQAP) [IEEE Std 730 1984], provide guidance to companies on how they should organize their quality management system.

This chapter is split into two sections, each of which describes a specific standard. The first section contains a discussion on ISO 9001, while the second contains a discussion on IEEE standard for SQAP. Both standards provide guidance to software developers on how to implement, maintain, and improve a quality system capable of ensuring high-quality software.

## 3.2. The ISO 9000 Series of Standards

The ISO 9000 series of standards is a series of international standard quality standards, developed by the international organization for standardization, that applies to the quality management system and the process used to produce a product. The ISO 9000 standards are based on the premise that if the production process is right, the product produced will be right. ISO 9000 establishes a basic set of quality system requirements necessary to ensure that the process is capable of consistently producing products that meet the expectation of the customer. ISO 9000 is a *quality system standard* [Schmauch 1994].

The standards describe *what*, at a minimum, must be accomplished-they do not specify *how* things must be done.

ISO 9000 raises quality awareness. If a company already has an established quality management system, it can compare its quality system against the ISO 9000 standards. If the quality system does not stack up well against the standard, serious thoughts must be given to improve the quality system. However, if the company does not have a formal quality management system, the ISO 9000 standards provide an excellent starting point and framework against which to define the quality system.

A well-designed, well-implemented, and carefully managed ISO 9000 quality system provides confidence that the output of the process will meet customer expectations and requirements.

ISO 9000 does this by requiring that every activity affecting quality be conducted in a three-part cycle: *planning*, *control* and *documentation*.

Activities affecting quality must be *planned* to ensure that goals, authority, and responsibility are defined and understood.

Activities affecting quality must be *controlled* to ensure that specified requirements at all levels are met, problems are anticipated and averted and corrective actions are planned and carried out.

Activities affecting quality must be *documented* to ensure understanding of quality objectives and methods, smooth interaction within the organization, feedback for the planning cycle, and objective evidence of quality system performance for those who require it, such as customers. To ensure that goals, authority, and responsibility are defined and understood.

The ISO 9000 series of standards contains three individual but related standards that apply to quality management and quality assurance. The ISO 9000 series of standards consists of five sections: ISO 9000, ISO 9001, ISO 9002, ISO 9003, and ISO 9004. The standards are specified by ISO 9001, ISO 9002, ISO 9003 with ISO 9000 providing guidance related to which standard to use and ISO 9004 providing amplification and guidance for implementing the standards. ISO 9001 covering processes encompassing product design/development, production, installation, and servicing. ISO 9002 covering processes encompassing production and installation. ISO 9003 covering processes encompassing final inspection and test.
Because ISO 9001 covers more aspects of development, more elements of the standard apply to ISO 9001 than to ISO 9002 and ISO 9003. ISO 9001 is a superset of ISO 9002 which in turn is a superset of ISO 9003.

### 3.2.1. Why be interested in ISO 9000: its Benefits

There are numerous benefits for having ISO 9000 conforming quality systems. The ISO 9000 series of standards provides an excellent basis against which to measure the quality system. Quality system should be judged against ISO 9000 to ensure consistent provision of quality products and service. First and foremost it is a matter of quality. Since quality products are our objective, then ISO 9000 should be of interest to us.
Benefits of ISO 9000 include [Schmauch 1994]:

- a foundation for quality products: an ISO 9000 conforming quality system will ensure that the development process has a level of control, discipline, and

consistency in building quality into products, thereby reducing rework cost. Continual improvement in the quality system, as required by the standard, will lead to continual improvement in the quality of the products.

- increased productivity and reduced cost: many companies that have already implemented ISO 9000 conforming quality systems are finding increases in development productivity. This actually makes sense. Doing the job correctly the first time under controlled, repeatable processes reduces the amount of rework and corrective actions required for products produced by less controlled processes. It reduces the amount of wasted time, energy, and money. It also reduces the amount of misunderstanding between and among various developers, which slows down the process.

- consistency: an ISO 9000 quality system will ensure the customers that not only quality products will be produced, but producing better and better products will be continual.

- improved competitiveness: conforming to the ISO 9000 quality assurance standards is becoming essential to succeed in an increasingly competitive global marketplace. ISO 9000 will indicate to customers that the products are more likely to meet claims made about them than products with similar claims made by non-ISO 9000 competitors. Also, ISO 9000 is almost a requirement to the market. European companies favor suppliers that have achieved or are in the process of achieving ISO 9000 registration.

- customers demand it: more and more, as ISO 9000 becomes more prevalent, customers are demanding it. The ISO 9000 movement is being driven by purchasers who are demanding more assurance that the products they purchase do what manufacturers claim.

- corporate image: ISO 9000 is being used to differentiate between quality companies and the rest of the world. This may be a valid distinction. ISO 9000 conformance requires a demonstrated continuing commitment to quality. A company with ISO 9000 conforming quality systems is a company that is committed to quality.

So, there are two broad ways to apply the ISO 9000 quality system standard. One way is to implement it for *quality management purposes*-that is to obtain its benefits for their own sake. The other way is to obtain a *certification* or *registration* to the ISO 9000 quality system standard. According to a study by Britain's authoritative for Quality Assurance, most firms seek ISO 9000 registration because of pressure from customers [Johnson 1993].

As a result, the achievement of ISO 9000 standards is a powerful strategic tool, whether the organization goes the certification route or not.

### 3.2.2. Standard Elements of ISO 9000

Since ISO 9001 is a superset of ISO 9002, which in turn is superset of ISO 9003, the focus will be on ISO 9001, because it is the most comprehensive of the standards and is mostly frequently applied to software development. The ISO 9000 series of standards specifies 20 standard elements. ISO 9001 requires conformance to all 20 of the elements, while ISO 9002 and ISO 9003 require conformance to 18 and 12 elements, respectively. The underlying essence of the ISO 9000 series is that the quality system must be documented, controlled, auditable, monitored, improved, and effective. In addition, management must be committed and employees must be involved. If the processes are documented, controlled, auditable, effective, continually monitored, and improved, and the management is committed and the employees are involved, then the requirements of the ISO 9000 series of standards are being met.

A summary of the elements of ISO 9001 will be given [Schmauch 1994].

*Management responsibility*

Management is responsible for establishing a quality policy and committing to it. The quality policy must clearly define responsibilities, authority, and interrelations of those with direct influence on quality. Management is responsible for communicating and making sure that people are aware of and understand the quality policy. This requirement pertains to all manufacturing environments, including software development, and emphasizes that quality must be a major objective and that quality goals and objectives must come from the top of the organization. Required verification activities need to be identified and resources (qualified personnel and money) must be provided for that activity. A manager who is responsible for the quality system must be assigned; and management must periodically review the quality system to ensure its continuing effectiveness. Assigning a management representative with authority to get things done is one way management can demonstrate its commitment to quality.

*Quality system*

A quality system, conforming with the ISO 9000 series of standards, must be established, documented, implemented, and maintained. A documented quality system must be in a place that enables the delivery of quality products. A quality manual is often used to document the quality system, and help to satisfy this element of the standards. Establishing and maintaining a documented quality system is a mean of ensuring that product conforms to specified requirements.

*Contract review*

Procedures must be in place, for the developer, to ensure that what is expected is adequately defined, documented and clear, and understood so the expected results can be produced.

*Design control*

Documented procedures for the design and changes to the design must be in place for controlling and verifying the design output to ensure that specified requirements will be met. Plans for each stage of development activity are required, planning and carrying out design reviews, assigning design verification to qualified personnel are also required. The intent is to avoid proceeding into production with a design that will not produce the expected results. In the world of software, it is quite often that the final product turns out to be something other than what was originally intended. Conformance to ISO 9000 standards, and this element and contract review, in particular, should help prevent this.

*Document control*

There must be defined procedures to control all documents, including review, approval, and change, and to ensure that the right level of information is available to the right people at the right time. This ensures that everyone on the project is working from the same level of document, such as specifications. Documentation that needs to be controlled, must be identified. This includes all internal documentation that affects the product and its quality. A master list of current documents must be maintained. The list would contain the names of controlled documents along with the name of each document owner, date of last update, review status, approvals.

*Purchasing*

If parts, used in the product or in the production of the product, are obtained from outside the organization, it must be ensured that they work as expected before using them. Subcontractors must also be selected based on their ability to produce what is expected from them. The ISO 9000 series of standards clearly places the responsibility for assuring the quality of the total product on the organization that delivers the final product. The supplier is responsible for the quality of all the parts of the product regardless of where or by whom they were produced. The intent of this element is to

ensure that the supplier validates all parts that go into the product when the parts are received.

*Purchaser-supplied products*

Procedures for verification, safe storage, and maintenance of products, or parts, provided by the customer to be included in the product are required. This applies when a customer's product is being modified. The supplier must verify that the product received is what is expected. This standards element requires the developer to verify that what was provided by the purchaser is what was expected. If the developer doesn't know what he/she expected or cannot verify it, how can he/she possibly plan what has to be done?

*Product identification and traceability*

Procedures for identifying and tracing the product during all stages of production, delivery, and installation are required. It is important to satisfy this element of the standard for the software development environment. Applying this element ensures that the developer controls the constituent parts of the product during development, and knows the exact content of products that have been delivered to customers so that the right service is provided to the customers who have different versions of the product.

*Process control*

It is important that the development process is carried out under controlled conditions, including, monitoring progress, approval processes and equipment. This is a key element in the standards. Several things need to be agreed upon. First, what is production? It is the set of activities that follows design completion and ends with product delivery. Second, what does it mean to be controlled? It is often easy to recognize when something is out of control while it is more difficult to show something is controlled. Control means that for all items relating to the product being

developed or the production of the product, there is an owner, with authority to make decisions and procedures for appropriate review, approval, and change. Third, this element of the standard makes provision for special processes. Until there are ways to produce zero-defect code, software will continue to fall under the category of special processes. Special processes are processes for which results cannot be fully verified by inspection and testing and the deficiencies become apparent after the product is in use. Special processes require that they be continuously monitored and that documented procedures be strictly followed. This achieved by step-by-step verification as the product progresses through various stages of the development process and with adherence to the documented processes and procedures.

*Inspection and testing*

Procedures for all levels of inspection and testing that have been identified as being required must be defined. Records of testing activities must be maintained. So, the testing that is required must be defined, documented, carried out, and it must be shown that the required testing was successfully completed.

*Inspection, measuring, and test equipment*

A demonstration that tools for testing, verification, validation, measurement can serve their intended purpose. Test tools must be validated and controlled.

*Inspection and test status*

During test stages, the test status of the various parts/product that are being developed throughout the process must be identified. Part of having the process under control is knowing and being able to show the status at any time.

*Control of non-conforming products*

Procedures for controlling a product that does not conform to its specified requirements must be established. Defects in software products can be discovered both

before and after the product is delivered to customers. Procedures for dealing with the product for which defects are discovered after it has been delivered are typically addressed in response to element, servicing. When defects are discovered in a product before the product is delivered, the most common practice is to rework the product until it passes all required testing.

*Corrective action*

Procedures for investigating the causes for non-conforming products and ensuring corrective actions must be established. It is worth noting that this element addresses corrective action to the *process* used to develop the product, not to the product. Procedures for corrective action to products that do not meet the specified requirements are addressed in control of non conforming products, and servicing. This element ensures continual monitoring, assessment, and improvement of the development process and quality system. Records of product defects and problems and customer complaints may be kept to satisfy this requirement. In addition, development process metrics need to be kept.

*Handling, storage, packaging, and delivery*

All parts and work items must be stored in a safe, secure, and controlled place. Procedures to ensure that the intended product is sent to the customer, and that the customer received what was sent must be established. The verification must be done prior to customer installation.

*Quality records*

Identification of whatever records needed to demonstrate and improve the effectiveness of the quality system. The standard does not specify what records must be kept. This involves identifying and keeping both product and process metrics.

*Internal quality audits*

Periodic internal audits of the quality system must be conducted by qualified personnel to determine the effectiveness of the quality system. The audits must be carried according to plans and documented procedures. These audits differ from ISO 9000 audits in that they must determine how effective the quality system is, which is not part of an ISO 9000 audit.

*Training*

Training needs must be identified, required training must be provided, and records of the training must be kept.

*Servicing*

Procedures for servicing the product, when it is specified in the contract, must be established. It requires documented procedures for correcting defects and non-conformities found in the product after it has been delivered to the customer.

*Statistical techniques*

Proofs must be provided to show that any metrics or measurements used during development of the product or to determine the quality of the product and the effectiveness of the quality system are correct and accurate. In addition to validating metrics, validity of any predictive algorithm used (e.g. for predicting number of remaining problems) and methods used to collect data is required.

## 3.3. IEEE Standard for SQAP

In this section, we introduce IEEE standard for SQAP [IEEE Std 730 1984], an internationally accepted standard which provides the basis of quality assurance plans. The purpose of this standard is to provide uniform, minimum acceptable requirements for preparation and content of SQAP's.

The organization or person responsible for SQA shall prepare a SQAP that includes the sections listed below, ordered in the prescribed sequence.

Additional sections may be added at the end, as required. Some of the material may appear in other documents. If so, then reference to those documents should be made in the body of the plan.

*Purpose* (section 1 of the SQAP)

This section states the specific purpose and scope of the particular SQAP. It lists the names of the software product items covered by the SQAP and the intended use of the software. It states the portion of the software life cycle covered by the SQAP for each software item specified.

*Referenced Documents* (section 2 of the SQAP)

This section provides a complete list of all documents which are referenced elsewhere in the text of the plan. It must also be stated where these documents can be obtained and who is responsible for them.

*Management* (section 3 of the SQAP)

This part of the plan describes the organizations, tasks, and responsibilities of the development process.

Organization

The organization created, influencing and controlling the quality of the software, is shown by means of an organizational structure diagram with additional written annotations. They contain a description of each group of the created organization which carries out quality assurance tasks; responsibilities which can be delegated; responsibilities for reports; identification of those groups which are responsible for the product release; identification of those groups which examine the SQAP; all procedures which may be invoked for solving conflict between

organizational groups; the size of magnitude of the quality assurance organization; and all deviations from quality policy formerly stipulated by the organization, or deviations form measures and standards for quality assurance. Organizational dependence or independence of the elements responsible for SQA from those responsible for software development shall be clearly described.

Tasks

All elements in this organization should be described in full detail, so that the tasks which are listed in the SQAP are allocated directly to the elements of the organization. Thus, the portion of the software life cycle covered by the SQAP, the tasks to be performed with special emphasis on SQA activities, and the relationships between these tasks and the planned major check-points shall be described.
The description of tasks in QA, in particular the sequence of the tasks must cover the entire software life cycle. This must include the names of the people publishing the SQAP and distributing, maintaining and releasing it.

Responsibilities

The description of responsibilities identifies which quality assurance groups are responsible for which quality assurance tasks.

*Documentation* (section 4 of the SQAP)
This part of the plan includes three sections, the second of which includes six sub-sections.

Purpose

This section identifies the documentation governing the development, verification and validation, use, and maintenance of the software. In addition, it states how the documents are to be checked for adequacy. That is, all reviews and audits are noted which comment on the suitability and quality of the documentation. It identifies

the reviews and audits by which the adequacy of each document shall be confirmed, with reference to section 6 of the SQAP.

Minimum Documentation Requirements

To ensure that the implementation of software satisfies the requirements, the following documentation is required as a minimum.

- *Software Requirements Specification (SRS)*

Software Requirements Specification (SRS) which clearly and precisely describe each of the essential requirements (functions, performance, design constraints) of the software and the external interfaces. Each requirement is defined such that its achievement is capable of being verified by a prescribed method, like inspection, analysis or test.

- *Software Design Description (SDD)*

Software Design Description (SDD) which describes how the software will be structured to satisfy the requirements in the SRS. The SDD describes the major components of the software design including databases and internal interfaces. The SDD shall be prepared first as the Preliminary SDD and subsequently expanded to produce the Detailed SDD.

- *Software Verification and Validation Plan (SVVP)*

Software Verification and Validation Plan (SVVP) which describes the methods (like, inspection, analysis or test) to be used to verify the requirements in the SRS have been approved by an appropriate authority, that the requirements in the SRS are implemented in the design expressed in the SDD and further into the code, and that the code, when executed, meets the requirements expressed in the SRS.

- *Software Verification and Validation Report(SVVR)*

Software Verification and Validation Report (SVVR) which describes the results of the execution of the SVVP. It includes the results of all reviews, audits, and tests required by the SQAP.

- *User Documentation*

User documentation, like manual and guide, specifies the required data and control inputs, input sequences, options, program limitations and other activities necessary for successful execution of the software. All error messages shall be identified and corrective actions described. A method of describing user-identified errors or problems to the developer is described. Software with no user interaction has no need for user documentation.

- *Software Configuration Management Plan*

The Software Configuration Management Plan (SCMP) documents methods to be used for identifying software items, controlling and implementing changes, and recording and reporting change implementation status. This documentation is either provided explicitly in this section, or refers to an existing SCMP.

Other

Other documentation may include Procedures Manual, and User's Guide.

*Standards, Practices, and Conventions* (section 5 of the SQAP)

This part of the plan includes two sections: purpose and content.

Purpose

In this section, all standards, practices, and conventions to be applied are identified. In addition, it states how compliance with these items is to be monitored and assured.

Content

The subjects covered shall include the basic technical, design, and programming activities involved, such as documentation, module naming, programming, inspection, and testing.

A minimum of standards, practices, and conventions must exist for requirements specifications, design, implementation (special coding and comments), testing and documentation. Selected SQA product and process metrics such as Branch metric, Decision point metric, Domain metric, Error message metric, Requirements demonstration metric must be provided.

*Reviews and Audits* (section 6 of the SQAP)

This part of the plan includes three sections, the second of which includes ten sub-sections.

Purpose

In this section, technical and managerial reviews and audits to be conducted are listed with the date of execution. It also states how these reviews and audits are to be accomplished. It states also what further actions are required and how they are to be implemented and verified.

Minimum Requirements

As a minimum, the following reviews and audits should be conducted.

- *Software Requirements Review*

  Software Requirements Review (SRR) which is held to ensure the adequacy of the requirements stated in the SRS.

- *Preliminary Design Review*

  Preliminary Design Review (PDR) which is held to evaluate the technical adequacy of the preliminary design of the software as depicted in a preliminary version of the SDD.

- *Critical Design Review*

Critical Design Review (CDR) which is held to determine the acceptability of the detailed software designs as depicted in the SDD in satisfying the requirements of the SRS.

- *Software Verification and Validation Plan Review*

Software Verification and Validation Plan Review (SVVPR) that is held to evaluate the adequacy and completeness of the verification and validation methods described in the SVVP.

- *Functional Audit*

Functional Audit that is held prior to the software delivery to verify that all requirements specified in the SRS have been met.

- *Physical Audit*

Physical Audit which is held to verify that the software and its documentation are internally consistent and are ready for delivery.

- *In-Process Audits*

In-process Audits of a sample of the design are held to verify consistency of the design, including: code versus design documentation, interface specifications-hardware and software, design implementation versus functional requirements, functional requirements versus test descriptions.

- *Managerial Reviews*

Management Reviews are held periodically to assess the execution of this plan. These reviews are held by an organizational element independent of the unit being reviewed, or by a qualified third party. This review may require additional changes in the SQAP.

- *SCMP Review*

The Software Configuration Management Plan Review (SCMPR) is held to evaluate the adequacy and completeness of the configuration management methods defined in the SCMP.

- *Post Mortem Review*

This review is held at the conclusion of the project to assess the development activities implemented on that project and to provide recommendations for appropriate actions.

Other

Other reviews and audits may include the User Documentation Review (UDR). This review is held to evaluate the adequacy, completeness, clarity, correctness and usability of user documentation.

*Test* (section 7 of the SQAP)

This section identifies all the tests not included in the SVVP for the software covered by the SQAP and states the methods to be used.

*Problem Reporting and Corrective Action* (section 8 of the SQAP)

This section describes the practices and procedures to be followed for reporting, tracking, and resolving software problems in both development and maintenance process. It also specifies who in the organization is responsible for the execution of these procedures.

*Tools, Techniques, and Methodologies* (section 9 of the SQAP)

This section identifies the special software tools, techniques, and methodologies employed on the specific project that supports SQA, state their purpose, and describe their use.

*Code Control* (section 10 of the SQAP)

This part of the plan defines the procedures, methods and tools used to maintain, store, secure and document already validated versions of identified software during all phases of the software life cycle. This can be done with the use of a Computer Program

Library. It may be provided as a part of the SCMP. If so, an appropriate reference shall be made.

*Media Control* (section 11 of the SQAP)

This section states the methods and tools to be used to identify the media for each computer product and the documentation required to store the media, including the copy and restore process, and to protect computer program physical media from unauthorized access or inadvertent damage or degradation during all phases of the software life cycle. This may be provided as part of the SCMP. If so, an appropriate reference shall be made.

*Supplier Control* (section 12 of the SQAP)

This section states the provision for assuring that software provided by suppliers meets established technical requirements. In addition, this section states the methods that will be used to assure that the software supplier receives adequate and complete requirements. For previously developed software, this section states the methods to be used to assure the suitability of the product for use with the software items covered by the SQAP. For software that is to be developed, the supplier is required to prepare and implement a SQAP in accordance with this standard. This section also states the methods to be employed to assure that the developers comply with the requirements of this standard.

*Records Collection, Maintenance, and Retention* (section 13 of the SQAP)

This section identifies the SQA documentation to be retained, states the methods and facilities to be used to assemble, safeguard, and maintain this documentation, and designates the retention period.

*Training* (section 14 of the SQAP)

This section identifies the training activities necessary to meet the needs of the SQAP.

*Risk Management* (section 15 of the SQAP)

This section specifies the methods and procedures employed to identify, assess, monitor, and control areas of risk arising during the portion of the software life cycle covered by the SQAP.

# Chapter 4

# Suggested Quality System

In this chapter a quality system is suggested based on the standards discussed in the previous chapter. The suggested program is split into sections each of which forms an item. A description of each item is provided to present an understandable quality system. The quality system is designed for a small to medium scale Lebanese company. It is designed to be effective for the company and as comprehensive as necessary to meet the company quality objectives not the standards. Unnecessary items are not included just to satisfy the standards. Conversely, if there are quality considerations that are needed, they will be included in the quality system though they may not be required by the standards.

The ISO 9000 series of standards was originally developed for the manufacturing environment. It is not written to any specific industry and is intended to be relevant to all types of business. Recognition that the process for development and maintenance of software is different from that of most other types of industrial products has lead to the fact that serious considerations are taken when designing quality systems for software.

The aim of the quality system is to ensure that the product meets the customer's quality requirements. It is based on the premise that associated with each software development project is a life-cycle consisting of a set of phases. In the proposed quality system, the life-cycle begins with the proposal phase followed by the requirement phase, planning phase, design phase, implementation, integration and test phase, and finally ends with the maintenance phase.

As mentioned earlier, software development is a very complex process which requires the use of many different disciplines for the development of a product to satisfy its

requirements. The necessary disciplines are Project Management, Quality Assurance, Configuration Management and Software Engineering for the implementation process. Since the purpose is to present the role of the SQA group in each activity in the quality system, the emphasis is on the QA discipline. However, to bring about a quality product, the elements of SQA must be coordinated with the elements of the other disciplines. For this reason, the planning component in project management, and the interrelationship of QA and CM, and Software Engineering are of importance. Throughout the chapter, necessary plans are presented.

Following is a presentation of the quality system.

## 4.1. Management Responsibility

This part of the system is concerned with the fact that the company should have a quality system, that management should support that system and that a very senior member of staff-designated management representative, Quality Manager (QM)-should be responsible for the system. Three requirements are essential for the success of the quality system.

### 4.1.1. Quality Policy

The company must define and document its quality policy, and objectives for, and commitment to, quality and customer satisfaction. The quality policy must be supported by the highest levels of management. A documented quality policy, signed by the highest level of management serves this purpose. However, it must be more than "lip service" by the person who signs it. The quality policy states the organization's quality goals and objectives and the strategy for achieving them, along with the facility's quality image and reputation. Every quality professional has asserted that the

success of a quality system is directly related to the consistency and intensity of top management's commitment [Schmauch 1994]. Management defines it quality policy and executes it through an organization of people and resources. Management also participates actively in the quality system by conducting verification and review activities. The quality policy must be publicized regularly so that it is understood at all levels of the facility. Quality policies do not have to be lengthy to be effective.

### 4.1.2. Organizational Structure

There are many ways to organize a software project. The larger the project the more critical the organizational structure becomes. Badly organized projects lead to confusion, and confusion leads to project failure.

Software project is the process of planning, organizing, staffing, monitoring, controlling, and leading a software project. Rarely can all these tasks be performed by the Project Manager (PM), in fact, they should not. The control and monitoring activities can be assigned to project support groups. These support groups not only disburden the PM and the development engineers from the support tasks, but they also perform these tasks better by concentrating their efforts on specific support functions. Three major technical support functions are required in every software development project: Configuration Control (CC) to manage the changes to the software product being developed, QA to monitor and control the quality of the product, and testing to verify compliance with the product's requirements.

The basic structure of a project in which below the PM are just two general functions: development and support is illustrated in Figure 4.1.

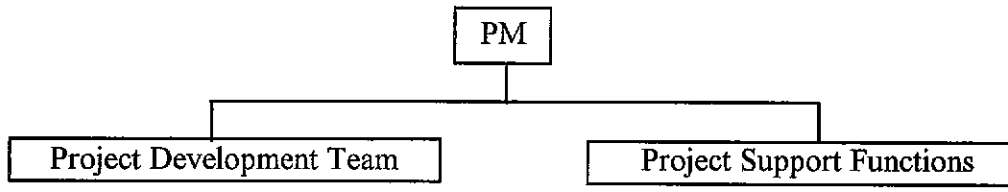


**Figure 4.1.** Project Basic Structure

This basic software project structure is valid for very small projects (up to five developers), though occasionally it can still be found in larger projects.

A detailed organizational chart including all major support functions is illustrated in Figure 4.2.

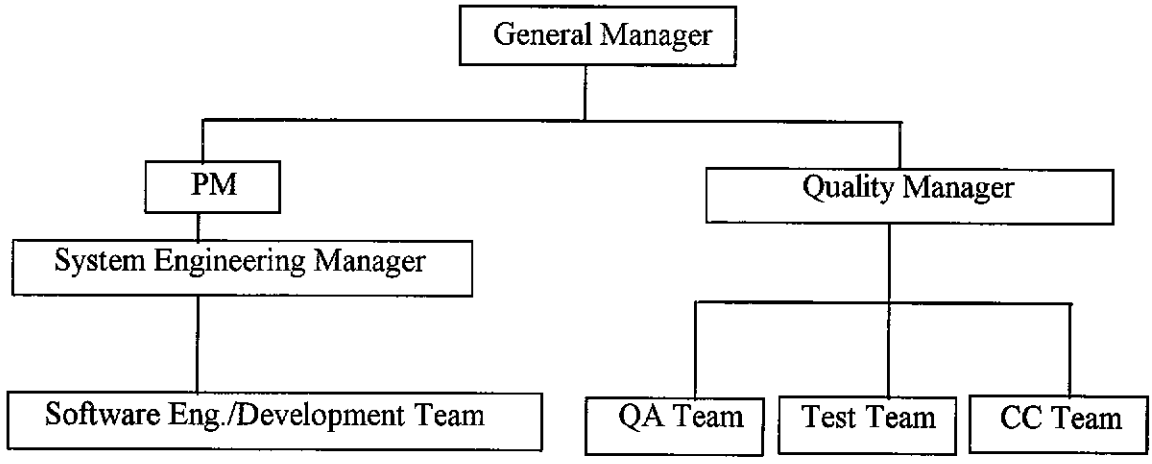


**Figure 4.2.** Detailed Organizational Chart

This organizational structure is suitable also for large projects (with a staff exceeding twenty). Smaller projects may not require a separate test team, CC and QA groups. Very large projects (exceeding a staff of forty) can often be managed more easily by dividing the project into sub-projects [Bennatan 1992].

The project's organizational structure is dependent on the type of project being developed. Some of the issues that must be considered are:

-Project size: the larger the project, the more important the organization. Large projects have significant human communications and coordination overhead, and therefore require more support functions.

-Hardware/software development projects: the simultaneous development of hardware and software is not easy. Planning, integration and testing are much more complicated, and require dedicated support groups.

-High reliability systems: any system is sensitive to issues of reliability require a major effort in QA and a separate QA organization. This includes military to life-saving systems and marketable software products.

-Corporate structure: the project's organization is largely dependent on the overall structure of the company within which the project is being developed. Many of the project support functions can be provided by centralized groups within the company. The size of a support group is clearly dependent on the size of the project; for example, a large project may require a group of two or three configuration control engineers, a medium size project may require one configuration control engineer and a small project may assign this task to a development engineer.

These decisions must be made by the PM during the initial stages of the project. Project support functions that are well planned at the start of the project will contribute to effective project management throughout the project.

#### 4.1.2.1. Roles, Responsibility and Authority

The responsibility, authority and the interrelation of all personnel who manage, perform and verify work affecting quality are defined.

*Project Manager* (PM) : directs, controls, administers and regulates a project to build a hardware/software system. He is responsible to the customer. Most of the people working on a project would report to the PM, although for some disciplines we might have a reporting relationship. The engineering groups would have an indirect reporting relationship to the PM.

*System Engineering Team* : is the collection of individuals (both managers and technical staff) who have responsibility for specifying the system requirements; allocating the system requirement to the hardware, software, and other components; specifying the interfaces between the hardware, software, and other components; and monitoring the design and development of these components to ensure conformance with their specifications. However, the emphasis is on the *Software Engineering Team.*

*Software Engineering Team* : is the collection of individuals (managers and technical staff) who have responsibility for performing the software development and maintenance activities (requirements analysis, design, code, and test) for a project. The technical people are the analysts, programmers, engineers who perform these activities, but who are not managers.

*SQA Team* : is the collection of individuals (managers and staff) who plan and implement the project's QA activities to ensure the software process steps and standards are followed.

*Software Configuration Management Team* (SCM/CC) : is the collection of individuals (managers and staff) who plan, coordinate, and implement the formal CM activities for the software project.

*System Test Team* : is the collection of individuals (managers and staff) who have responsibility for planning and performing the system testing of the software, whether

independent or not, to determine whether the software product satisfies its requirements.

It is the PM's responsibility to organize the project support groups and to document their planned activities in the Project Development Plan (PDP). If these groups already exist within the organization, then their support needs to be coordinated and scheduled for the project. If the groups do not exist then they must be established within the project development team.

Responsibility must always go side by side with authority. All project staff members must be given authority to act within their area of responsibility. The QM must have the authority to approve or reject product components, and development engineers must have the authority to make design decisions related to the components that they are developing. However, no authority is absolute, and higher level personnel within the project must constantly review these decisions and step in when they feel an error is being made.

#### 4.1.2.2. Independence in the Organizational Structure

As to the organizational independence, the company must take care of the functions that call for *organizational* independence. Should organizational independence be needed, the following two disciplines are involved: the SQA group and the test group.

For small projects, the support functions can be combined together. Organizational independence is not needed. Merging these functions means assigning the responsibilities to the same person. However, independence of the support groups *within* the organization is required.

The SQA group has a reporting channel to senior management that is independent of the PM. The independence of the SQA group is necessary so its members can perform

their jobs without being influenced by project schedule and cost pressures. Internal independence provide the individuals performing the SQA activities with the organizational freedom to be the "eyes" and "ears" of senior management on the project, protect the individual performing the SQA activities from adverse personnel actions when there is a need to escalate deviations outside the project and provide senior management with confidence that objective information on the process and products of the project is being reported.

The test group prepares and plans the system and acceptance test cases and procedures. Such independence ensures that the testers are not appropriately influenced by the design and implementation decisions made by the software developers or maintainers. Test procedures and testing are always best when conducted by a separate team. The decision on whether supervision of testing activities can be assigned to SQA depends on many factors, including the independence of the SQA team, the size of the project and the complexity of the project. When testing is performed by an independent test team, SQA's involvement will be minimal. In most cases it is the responsibility of the SQA team to plan and supervise the testing of the system.

While test teams function primarily during the integration and test phases of the development cycle, the QA team functions throughout the project's development cycle.

So, it is undesirable for SQA to be performed by a member of the development team. However, small organizations and projects often cannot justify the cost of a dedicated SQA group. This problem can be solved by having a single SQA engineer responsible for two or three small projects.

The SQA engineer or QM activities cover the review and approval of the development methodology, evaluation and selection of development tools, the software

documentation, the planning, supervision and approval of testing, and the administration of CC.

Merging SQA and CC is not uncommon, and both responsibilities may be assigned to the same person.

### 4.1.3. Management Review

It is important that the quality system be subject to continual review and assessment of its appropriateness and effectiveness. To attain required quality, management has to specify its objectives, establish plans and procedures to accomplish them, assign duties, delegate authority, set up adequate methods and standards, and evaluate results objectively. In order to ensure that these requirements for quality continue to be met, the management should periodically and systematically conduct formal reviews. The findings of the quality system reviews should be analyzed to determine the actions to maintain the system's effectiveness.

Whenever problems are not identified at their earliest point by the quality system, appropriate remedial actions must be taken. Changes to the software development environment leads to a re-examination of the quality system. Such changes include changes in the hardware platform or system software, changes in the software development platform (new language, CASE tools), changes in business direction affecting the software development function, and changes to internal software development standards and procedures.

Such changes are documented then discussed with developers and agreed upon prior to implementation.

## 4.2. Quality Planning

The second element of the quality system is quality planning.

The SQA person works with the software project during its early stages to establish plans, standards, and procedures that add value to the software project and satisfy the constraints of the project and the organization's policies. By participating in establishing the plans, standards, and procedures, the SQA person helps ensure they fit the project's needs and verifies that they will be usable for performing reviews and audits throughout the software life-cycle. He reviews the project activities and audits software work products and provides management with visibility as to whether the software project is adhering to the established plans, standards, and procedures. Compliance issues are addressed within the software project and resolved if possible. Non-compliance issues that cannot be resolved within the software project are addressed by senior management.

In order to meet quality requirements, many activities are required of the PM and the QM at the commencement of a new project, and throughout the project life-cycle. Quality planning is performed in two phases: initial planning and final planning.

Since SQA must play a lead role from the beginning of each phase, the management of each phase must be weighted equally in terms of importance to achieve a final quality product. No phase must be considered lightly or be weighted more than any other relative to the final product. If management accepted each phase in terms of what each phase must accomplish and directed efforts within each phase to accomplish its specific objectives, then no one phase would receive preferential treatment, or inadequate attention.

## 4.3. Initial Planning Phase

Management's first chance at planning an effective role for SQA is during the initial planning phase, also called proposal or concept phase, where the objectives of quality must first be defined. Defining quality objectives is particularly important to software quality and must be specifically and uniquely tailored to the project. In this sense, the what and how of SQA must be related to developmental processes and described for each phase of the software life cycle.

The proposal phase, then, is management's (both development and quality) first look at the plans and objectives for development. This is not a phase whose only objective is winning contracts. How tasks will be performed (process quality) and what the end products must do (product quality) are being firmly established in a legal and contractual sense.

It is important to point out that there is the case where no contract is involved, or at least only some form of informal agreement is in existence. This normally occurs when the computer department of a company produces a software system for another part of the company. While the business trend is towards internal contracts, the computer service departments are still producing software without any formal notion of a contract.

The customer should cooperate with the organization developing the product to provide all necessary information in a timely manner and resolve pending items. The customer assigns a representative with the responsibility for dealing with the developer on contractual matters-when the project is contracted out.

The customer's representative authority include defining the customer's requirements to the developer, answering questions from the developer, approving developer's proposals, concluding agreements with the developer, ensuring the customer's

organization observes the agreements made with the developer, defining acceptance criteria and procedures, and dealing with supplied software items that are found unsuitable for use.

Several items on quality are found to be relevant in the contract, or any other document produced at the end of this phase-initial version of the PDP. These items include the acceptance criteria, handling of changes in customer's requirements during the development, handling of problems detected after acceptance including quality related claims and customer's complaints, activities carried out by the customer, especially the customer's role in requirement specification, and acceptance, facilities, tools and software items to be provided by the customer, standards and procedures to be used. The QM should be an active participant during this phase to provide insight based on his previous experience into preventing problems, defining major risk areas, defining the way review processes and inspections are to be conducted, defining the need and use of procedures, and assessing how to modify existing procedures to fit a project. This is not intended to provide a complete set of SQA activities during the proposal phase but to emphasize the need for SQA personnel to be active participants in defining and proposing what must be accomplished and how these developmental activities will be processed to properly coordinate the planned activities of SQA with all the concerned organizational areas [Bennatan 1992].

The contract or any other documentation is reviewed at a meeting of representatives from quality and project management.

As a result, this phase provides the basis for the definition of the software requirements, and initial planning and preparation of estimates. It serves as an early version of the PDP. The quality aspects that are considered in this phase are often conducted informally. A description of the PDP is given in the following phase since this phase is not formal.

## 4.4. Final Planning Phase

Although the initial phase is not a mandatory formal development phase, and is often conducted informally, it is a phase during which the need for the software system is determined and the basic concept of the software system evolves. Following the initial phase is the final phase. This phase includes all the mandatory phases of software development. The following sections describe the management issues affecting quality associated with each phase, and the quality activities related to project elements.

### 4.4.1. The Software Requirements Phase

The software requirements phase is the first formal mandatory phase of software development. This phase provides a detailed description of the software system to be developed. It is according to the requirements specification that the software is tested at the end of the project to demonstrate that the required product has indeed been produced. The requirements specification answers the question *what* while attempting to avoid the question *how*.

The requirements phase forms the basis for the first major system baseline, the design of the system, and the Acceptance Test Procedures (ATP).

This phase produces one main product document that is the software requirements specification document, and two project planning documents, the PDP, and the software test plan.

This phase is perceived as the most difficult and important phase of the software development cycle. There is a basic conflict of interest between the customer and the management since the customer is reluctant to finalize the requirements because of the knowledge that once this is done any further changes may be costly and the management needs to finalize the requirements as soon as possible because progress will be slow as long as the product is not fully defined, and this is costly.

Several problems are encountered during this phase. Such problems include the frequent changes of requirements, closing requirements which is not always easy and requires experience, patience and firmness. It is the PM's job, to be resolute and decisive in requiring all parties to sign off the requirements for the project, because this phase cannot be completed without a formally approved requirement document. Another problem encountered during this phase is the staffing because locating suitable development team members can be a difficult task. It is always a problem for the PM. Assignment of team members should be completed during this phase. Equipment procurement, and binding estimates should be completed before the end of this phase. Staffing and equipment problems impact the estimates and the schedule in the PDP. All these problems delay the project.

Since the SQA group must play a role right from the start of the development process, after the customer has established the system requirements, the engineering group will perform checks-assessments-as to their adequacy. The QM uses the assessments to revise the process of establishing requirements, change methodologies, or enforce the procedures originally planned. Prototypes could be used to test the requirements, any vague or suspect requirements should be deleted.

The requirement phase formally concludes with the project's first major review: Software Requirement Review (SRR). This review declares the requirement document as the first approved project baseline. During the review, representatives of the requirements team are present, as are representatives of the customer organization. The meeting is usually chaired by the QM. The aim of the review is to ensure that the requirements are correct. The reviewers go through the specification document, ensuring that there are no misunderstandings as to what is meant by each statement in the document.

Obtaining formal approval for the requirements specification is the most difficult task of this phase. The requirements must evolve gradually. Several draft versions of the specifications might be submitted until convergence to the final approved document. As to the PDP, an initial draft is prepared, and it is updated throughout the life of the

project to reach a final PDP. Since the PDP is one of the requirement phase outcome, and although it progresses in parallel with the requirement phase, it is presented in the following separate phase, the planning phase.

### 4.4.2. The Project Planning Phase

During the planning phase, the software PDP is drawn up. It is one of the first formal documents produced by the PM. Within this document, the PM describes in detail how the project will be developed, what resources will be required, and how these resources will be used. A major aspect of the plan is estimating the duration of the project and how much it will cost. In other words, the project schedule is one of the most important parts of the PDP.

The following is a brief outline of some of the subjects covered in the PDP [Bennatan 1992]:

-*System overview*: a general overview of the project is included in the first section of the document.

-*Software development management*: this section includes four sub-sections. It describes the *project organization and resources* that will be used to develop the product, the organization of the *development facilities* that will be used to support the development effort, the *project organizational structure*, and the *personnel*.

-*Schedule and milestones*: this section provides the necessary information to prepare the development schedule. It includes the *scheduled activities*, *milestones and baselines*, and *budget administration*. It also involves a schedule of planned major QA activities in relation to project milestones.

While the schedule provides answers to two basic planning questions: *what* and *when*, the remaining sections discuss *how*.

The discussion in the *how* sections provides information on: *risk analysis*, *security*, *interface with external sources*, *procedure for reviews*, *corrective action process*, *standards*, *development methodologies* used, *testing*, and *SCM*.

Certainly, not all subjects are applicable to all projects. For example, the interface with external sources involves activities such as interfacing with subcontractors, and vendors. Some projects do not require this activity.

An initial PDP is prepared. It is further refined as the project goes through progressive stages.

The QM meticulously checks every aspect of the plan, and pays particular attention to the duration and cost estimates on which the plan as a whole is based. One way to do this is for management to obtain two or more independent estimates of both duration and cost at the start of the planning phase, and then to reconcile any significant differences. With regard to the document, an excellent way to verify it is by means of a review similar to the review of the specification document, conducted by the QM.

If the duration and cost estimates are satisfactory, then the customer gives the permission for the project to proceed. The next phase is to design the product.

### 4.4.3. The Design Phase

During this phase, the requirements are analyzed and the method of implementation is determined. Just as the requirement phase addressed the question *what?*, the design phase addresses the question *how?* The response to this question is documented in the software design specification document.

This phase provides the basis for the second major system baseline, the implementation of the system, and an updated development plan. It produces the following documents:

design specification (for large projects: top level design specification and detailed design specification), integration plan, and test case specification, describing in detail each individual low level test.

The design phase can be or is often divided into two separate phases: top level design and detailed design.

The design specification document establishes the project's second major baseline. In the case of two design phases, the detailed design specification is regarded as the major design baseline, and the top level design specification is regarded as a secondary baseline.

At the end of this phase, many of the project's unknowns become known, thus providing a significant improvement in the PDP estimates. Various project development parameters, such as the integration schedule and resources and the actual test cases for the test phase, can now be planned. The updated development plan can therefore be regarded at this stage as being significantly more reliable. In parallel with the design of the system, the development and integration platforms are installed including all the equipment required for system development and integration, estimates are significantly improved, project risk analysis is reviewed and updated, the project development schedule is updated. All these activities are included in a new major revision of the PDP [Bennatan 1992].

The design phase concludes with the sign-off of the design specification document. This usually occurs at a formal design review, referred to as the critical design review (CDR). If an intermediate top level design specification is prepared, then this document is signed-off at an initial preliminary design review (PDR). In view of the technical nature of most design documents, it is not usual for the customer to be present.

Members of the design team and the SQA person work through the design as a whole as well as each separate module, ensuring that the design is correct. The types of faults to look for include: logic faults, interface faults, lack of exception handling (processing of error conditions), and most important, non-conformance to the specifications. In addition, the review team should always be aware of the possibility that some specification faults were not detected during the previous phase.

The SQA person must carefully look for contradictions, ambiguities, and incompleteness. He ensures that the specifications are feasible (any specified hardware component is fast enough, or that the customer's current on-line disk storage capacity is adequate for handling the new product), he ensures that the specification document is a true reflection of the customer's requirements.

A critical aspect of testability is traceability. This means that every part of the design can be linked to a statement in the specification document. The SQA person will go through the design and have powerful tool for checking not just that the design is in accordance with the specification document, but that every statement of the specification document is reflected in some part of the design [Ince 1994].

This phase is characterized by delays because many changes in requirement and design are introduced, owing to late ideas, unfeasibility of requirement, additional new information, and by confusion because the team grows rapidly, the project hierarchy and responsibilities may not yet be clear. The design phase for the PM is a period of organization during which the project team structure is finalized and the assignment of responsibilities is completed. These tasks must be completed by the PM before the end of the design phase, as the confusion that may have accompanied the previous project phases cannot be carried over into the implementation phase.

When the design phase concludes with a formal review, the customer shares the responsibility for the design.

Although the Implementation phase and the Integration phase must be carried out in *parallel*, but since the purpose here is to present the role of SQA in each activity, we will separate them.

### 4.4.4. The Implementation Phase

During this phase the software modules are coded and initial unit tests are performed. Unit testing is carried out by the programmer on each individual module immediately after it is coded. The modules are run against test cases. The modules are then approved by software quality control and submitted to configuration control, which then releases modules for integration. While the programmer performs the informal testing, the QA group tests the modules methodically.

A detailed and well-structured design specification leads to relatively smooth and straight forward coding. In addition to running test cases, a code review is a powerful and successful technique for detecting programming faults. Here, the programmer guides the members of the review team through the listing of the module. The review team must include an SQA representative. The procedure is similar to reviews of specifications and designs.

The implementation phase links the design phase and integration phases of the system, and usually overlaps significantly with each of these other two phases. Overlapping will often occur when many parts of the system design are completed relatively quickly, leaving some of the design issues open for quite a while. Overlapping can shorten the development schedule significantly.

Overlapping of the design and implementation phases requires great care in assuring that only design-complete modules are approved for early implementation. There is the

risk that any later changes to the design of these modules may require re-coding, thus wasting resources. There is also the risk of the design being changed. With good planning and configuration control, these problems can be overcome.

On the other side, overlapping of coding and integration is less risky, and if planned correctly, can be an excellent time saver. The order of implementation of the modules should be well-planned to assure that they are released in the order required for integration.

The implementation phase includes the following activities: the development of the software code, preparation for integration and test of the system (next phase), the development of the maintenance phase, risk situations may materialize and actions taken according to plans, and the PDP is reviewed and updated.

The documents that are developed during this phase include, documenting coding decisions, unit tests and resolution of implementation problems, maintenance plan and documentation including all necessary documentation needed in system maintenance, initial versions of the user documentation, including reference manuals and operator guides.

During implementation, the atmosphere is characterized by the pressure to get going and show something, conflicts with the SQA and CC. These conflicts are due to increased involvement of these supervisory organizations or groups and their role in enforcing standards and orderly development procedures, and increased development team activity as the delivery dates become close.

In general, the implementation phase is a transitional period from specification to building. The atmosphere is dependent on the success of the previous specification phase and the expected success of the following integration and test phases. Contribution factors include inadequate requirements definition, insufficient resources

assignment, insufficient development time, some unresolved technical problems, and lacking of management support.

If the design phase is implemented well, most technical problems should be resolved by the end of design. If this is not so, confusion may follow due to the necessity to program and to solve technical problems simultaneously. This situation fall into the code and fix method.

### 4.4.5. The Integration and Test Phase

The purpose of integration testing is to check that the modules combine together to achieve a quality product that satisfies its specifications. During integration testing, particular care must be paid to testing the module interface. This checking is performed by the SQA person.

When the integration testing has been completed, *product testing* is performed by the SQA person to ensure a successful acceptance test. The functionality of the product as a whole is checked against the specifications. First, black-box test cases for the product as a whole are run. Second, the robustness of the product as a whole is tested. In addition, the product is subjected to stress testing, that is making sure that it behaves correctly when operating under a peak load. It is also subjected to volume testing, making sure that it can handle large input files. Third, the SQA person checks that the product satisfies all its constraints. For example, if the specifications state that the response time for 95% of the queries when the product is working under full load must be under three seconds, then it is the responsibility of the SQA person to verify that this is the case indeed. All these constraints will, of course, be checked during acceptance testing, and if the product fails to meet a major constraint during the acceptance testing, then the development organization will lose a considerable amount of credibility. Fourth, the SQA person reviews all documentation that in terms of the

contract is to be handed over to the customer together with the code. The SQA person checks that the documentation conforms to the standards laid down in the PDP. In addition, the documentation is verified against the product. For instance, the SQA person has to determine that the user manual indeed reflects the correct way of using the product.

Once the SQA person assures management that the product can handle anything the acceptance testers can throw at it, the product is handed over to the customer organization for acceptance testing.

The *acceptance testing* is the final aspect of integration testing. The purpose of the acceptance testing is for the customer to determine whether the product indeed satisfies its specifications as claimed by the developers.

Acceptance testing is done either by the customer organization, or by the SQA person in the presence of customer representatives, or by an independent SQA group hired by the customer for this purpose. The four major components of acceptance testing, namely testing correctness, robustness, performance, and documentation, are exactly what is done by the developer during product testing. This is not surprising, because product testing is a rehearsal for the acceptance test.

Acceptance testing is done on actual data, rather than test data. No matter how careful the development team or the SQA person might be, there is a significant difference between test cases, which are artificial, and actual data.

A software product cannot be considered to satisfy its specifications until the product has passed its acceptance tests.

The integration and test phase is the most difficult and the most important to plan. Any schedule delays at this point can be critical. It is important for the PM to produce the initial version of the system as soon as possible, since management has seen little to justify their investment. Conflicts with the customer occur since in this phase the customer can see an initial version of what the product will look like. Different

interpretations of the requirements emerge and need to be resolved by higher management level. Integration problems may require to return to the design phase which might lead to frustration.

Many of the pressures can be avoided by assuring good design, an efficient module coding plan, a well-organized development team, and a good integration plan. Many unexpected problems are encountered during this phase since the events described earlier in the project as risks now materialize. These include last minute failures due to design and implementation problems which emerged, third party problems, including late delivery from vendors and defects in their subsystem, last minute changes problem exists in all phases, and changes become more costly, budget overruns problem is derived from changes and design errors and project planning errors, staff motivation problem which occur toward the end of the phase, and project acceptance problems since the conflicts arise concerning the completion of the project. Many of these problems can be avoided by preparing for them early in the project.

### 4.4.6. The Maintenance Phase

The IEEE definition of software maintenance includes the correction of faults that existed in the software before its delivery, as well as changes to improve performance or to adapt the product to a changed environment [Wallmüller 1994].

Two aspects to the testing of changes to a product in operations mode that must be done by the SQA person before a product is distributed. The first is checking that the required changes have been correctly implemented. The second aspect is ensuring that, in the course of making the required changes to the product, no other inadvertent changes were made. Thus, once the SQA person has determined that the desired changes have been implemented, the product must be tested against previous test cases to make certain that the functionality of the rest of the product has not been

compromised. This procedure is called *regression testing*. To assist in performing regression testing, it is necessary that all previous test cases be retained, together with the results of running those test cases.

Moreover, the documentation must be updated to reflect the changes made. All these steps must be followed as a result of corrective maintenance which is initiated by a fault report, or as a result of perfective and adaptive maintenance which are initiated by a change in the requirements [Wallmüller 1994].

Testing during the maintenance phase is both difficult and time-consuming, and the SQA person should not underestimate it. Once the new version has been approved by the SQA person, it can be distributed.

Modifying software includes all characteristics of developing software. It includes all the phases of a full development project. Many of the development problems prevalent during the basic development phases are common in this phase. Modifications need to be described, designed, implemented, integrated and tested and budgeted.

Configuration control is important to manage the various changes to software, and control many releases and versions of the system.

Project management is not always carried over from the development of the software product to the maintenance phase. Maintenance requires a smaller team and a different type of management.

## 4.5. Quality Controls

The third element is quality control.

The determination of quality should not be postponed until development is complete. Effective software quality control requires frequent assessments throughout the development cycle. Thus, effective quality control coupled with a good requirements specification clearly increases the quality of the final product.

At this stage, the PM has identified the quality factors that are important during planning. Quality controls are needed to check that particular quality factors are

present in the system. This is achieved using analytical QA measures. Important aids to this element are quality tests. The quality tests are divided into two categories: static and dynamic tests.

### 4.5.1. Static Tests vs. Dynamic Tests

While static tests include reviews and audits, dynamic tests include execution of a program to test whether it behaves in accordance with the specification.
Although the activity to create a test is considered as a powerful bug preventer, the review and audit checklists are an even more powerful bug preventer than software testing, and they are the primary tool of the SQA effort. System test design cannot really begin until the first stage of the implementation phase, at this point there will be sufficient product detail available to allow the drafting of the test plan. Life cycle reviews and audit checklists, in contrast run the entire length of the product cycle, from initial conception through completion [Wallmüller 1994].

### 4.5.2. Reviews vs. Audits

Before looking in detail at the reviews and audits, we should make a brief distinction between their natures.
Reviews are designed to check the development of the software system and to make sure that the steps which have been, and are to be taken, and the standards outlined for them, are in accordance with organization policies, procedures, and all system requirements. Reviews are dynamic, in the sense that they deal with the on-going development of the software system, and preventive, in that they aim at stopping problems before they appear as bugs in the software product [Wallmüller 1994].
Audits, in contrast, are designed to check the state of the system being developed at a certain point, and to ensure that the current product is in accordance with organization policies, procedures, and all system requirements. Audits are static in that they deal

with the past development of the system leading to the current software product, and corrective, in that they aim at correcting any bugs currently in the software and any problems actually present in the supporting materials [Wallmüller 1994].

### 4.5.3. Reviews

From the planning phase, several documents are produced. At the end of each phase, a review is conducted by the QM using checklists relevant to each document and phase. Project reviews are where major project development decisions are finalized. These critical decisions are documented in the development specification documents, and are referred to as baselines. They become the primary source of reference for further development of the software product. As a minimum, the following reviews should be conducted.

#### 4.5.3.1. SRS Review

Inadequate requirements delay projects, and can cause their termination. An important aim during review of the SRS is the testing of whether certain quality features are met. The basic characteristics of a good SRS are: unambiguity, completeness, verifiability, consistency, modifiability, traceability, and usability during the operations and maintenance phase.

These characteristics are tested by means of a checklist which contain the following questions:

Is each requirement clear and does it have the same interpretation to all who read it? Are all requirements documented, assuring that no verbal understanding remain? Can we prove that each requirement has been met? Does any requirement conflict with any other requirement? Is the requirements specification documented in a way that enables it to be easily corrected or changed later? Are the origins of each requirement clear, and can the testing and design documents be later traced to requirements? Has the

requirements specification been written so that it can be understood not only by the organization writing it, but also by the software maintenance organization? Are the specified response times realistic? Are all necessary hardware resources specified? Have all the functions that the user needs been identified and specified? Has the acceptable level for accuracy been specified for results? Are the requirements comprehensible for those who have to design the project?
There are tools that have test functions to support the testing of the above questions.

#### 4.5.3.2. Design Reviews

Depending on the type of design, there is a distinction between high-level design reviews (PDR) and detailed design review (CDR). Design reviews have the following aims: determination and evaluation of the respective state of a design (completeness of feature), and discovery of errors and contradictions (contradictions between specification and design, or between module interfaces).

Checklists exist for PDR and CDR related to performance, user interfaces, data, functionality, documentation, standards, and syntax of design description. The checklist includes questions such as:
Are there any hints of non-fulfillment of performance requirements? Are the screen layouts not overloaded with information? Are the screen outputs clear? Are the layouts of the user interfaces uniform? Has user input been kept to a minimum? Are there missing or unused variables in a module? Are there missing or erroneous data types in a module? Are logical conditions non-existent, superfluous or faulty in a module? Is the design description incomplete, ambiguous? Are the algorithms in a module clearly specified? Has the syntax of the design notation been applied incorrectly? Does the design description contain spelling errors? Has a standard chosen not been adhered to?

#### 4.5.3.3. Post-Operation Review

The Post-Operation Review (POR) is accomplished during the operational and maintenance phase, after the Physical Audit (PA), and not sooner than ninety, nor later than one hundred and eighty, days after the delivery of the software product to the user [Vincent 1988]. Prior to the POR, the system designers and development team members will place the software system in operation, and the test team will perform any supplemental site tests required by the test plan. During the POR , the system designers will review and interpret the results of the supplemental site tests, and explain any problems encountered in the testing and the way to solve them.

Among the primary objectives of the POR are these: to verify the adequacy and completeness of system support manuals, to verify the adequacy of management plan, and its proper implementation and use, to ensure that all supplemental site testing is carried out completely and accurately, to verify that all test documentation is properly prepared to ensure that all problems, deficiencies discovered during testing have adequate solutions proposed and implemented, to ensure the continued quality of the software product through continued quality review, and the monitoring of problems and corrections, to ensure the maintainability of the product through a monitoring of design changes. Upon approval by the QM, the Operational Baseline(OB) is established. This baseline is set at the conclusion of the development cycle and finalizes the development of the software product.

### 4.5.4. Audits

In connection with quality, two kinds of audits can be differentiated: Functional Audit (FA), and Physical Audit (PA).

#### 4.5.4.1. Functional Audit

The FA is conducted after product testing is run. It is primarily concerned with the actual functioning of the software product, ensuring that all required fixes have been, or are being, implemented, that the testing required to this point was properly conducted, and that test results indicate that the software product meets performance specifications and requirements, including those required software quality factors [Vincent 1988].

Among the primary objectives of the FA are to ensure that major system modules are properly linked and tested, to verify that the test, as conducted, represent an adequate functional and system level test, to verify that all test documentation is properly prepared, to ensure that all problems, deficiencies discovered during testing have adequate solutions proposed and implemented, to ensure that all requirements specifications as outlined in the documentation are actually met. Upon approval by the QM, the Product Baseline(PB) is established.

#### 4.5.4.2. Physical Audit

The PA is accomplished during the operation phase of the life cycle, after AT is run. The primary intent of the PA is to verify that all products are in compliance with organization standards, policies, and procedures, and to ensure that these reflect required user, and quality standards. However, its emphasis will be on ensuring that the software and its documentation are internally consistent, and that the software product is ready for delivery [Vincent 1988].

Among the primary objectives of the PA are to verify that the AT, as conducted, represent an adequate system and integration test, to verify that all test documentation is properly prepared, to ensure that all problems, deficiencies discovered during testing have adequate solutions proposed and implemented, to ensure that all modifications made to the product are in accordance with the specifications and requirements settled

in the PB to ensure that all system manuals are in good order, and in conformance with the final product configuration, to ensure that all support programs and hardware are in good order, and in conformance with the final product configuration, to ensure that all implementation plans are in good order, and are acceptable to the user, the quality and project management. Once the QM has given its approval, the software product will be released to the user for initial operation.

### 4.5.5. Testing

Testing requires a considerable amount of effort. During testing, the test object is executed. A program is tested with a selection of input values. This serves to test whether the program behaves in accordance with the specification. Testing is one of the most important verification processes.

Four testing activities are required: unit testing, integration testing, system testing, and acceptance testing.

The responsibility of the QM lies in the execution of certain tasks related to testing. These include the creation, and maintenance of test plans, the organization of testing activities, the development of test specifications and test procedures, the creation and maintenance of test cases, the creation and maintenance of test documents, the procurement of test tools and aids, the execution of test reviews, and the testing of post-release changes.

A test process exists to carry out each of the testing activities. It consists of a certain action with its corresponding result.

#### 4.5.5.1. Test Planning

With this activity, a document is created which records the goals, the size, the method, the resources, the schedule and the responsibilities for the intended tests. This

document is the test plan where important features for the quality of testing are identified.

Each test must have quantifiable aims since testing without set goals is a waste of time and money, test goals must be quantifiable, each test case must be repeatable, for each test case there should be exact instructions for execution and evaluation, for each test case there should be an expected test result, testing must be conducted economically since a test object cannot be tested for all possible inputs.

During test planning, test goals are defined, and attempts are made to reach them through the selection of suitable test cases. Determining test cases makes test methods necessary. A test goal is a measurable metric which is used for assessing the results of a test. They enable an objective test procedure and facilitate management of the tests. Two test methods exist for the derivation of test cases: black-box and white-box. The most important black-box methods are function-coverage, equivalence class method, boundary value analysis, and the cause-effect graphing method. As to the white-box methods, there exist test methods based on coverage metrics such as statement coverage, branch decision coverage, condition coverage, coverage of all combinations of conditions and path coverage, and there exist test methods based on complexity metrics like those of McCabe such as cyclomatic complexity, essential complexity and actual complexity [Wallmüller 1994].

Not only test goals are defined but test termination criteria are also set during test planning. Two ways of terminating the test process. The first method is through a given number of errors which had to be found, the second is through observation of the error rate in the course of the testing phase.

Good test planning leads to tests which result in trust in the test object and a test process with a high probability of problem and error discovery. Test planning can begin as early as the requirements phase. The planning of system testing can take place after the release of requirement specification. The planning of integration testing can begin at the design stage. Module testing in the detailed design phase (module design)

is only function-oriented (black-box method), and test case design for white-box tests is carried out during the implementation of the module [Wallmüller 1994].

#### 4.5.5.2. Test Design

In the test design, detailed instructions related to test methods are given. It must be stated clearly how the goals contained in the test plan are to be reached. It specifies which methods are to be applied and in which way. A test design would specify what test software is needed, and what hardware base is needed. It also defines which test objects in which tests and on the basis of which criteria to be used to come to a decision on whether the test object does or does not pass the test. The test design gives rise to test specifications and test procedures.

#### 4.5.5.3. Test Case Determination

Test cases are specified based on the test design. It states in detail which test object in the execution of each test case is to be fed with what input, and what output should be generated.

#### 4.5.5.4. Test Procedure Planning

With this activity, the steps for the actual execution of the test are defined. The requirements for the execution of the test runs, the sequence of the test runs, and records and termination tasks for each test run are specified in detail. The output is a test procedure document.

The test case specification and test procedure specification documents can be combined together.

#### 4.5.5.5. Test Execution

The planned test runs are executed and their associated test log is established. All incidents and problems which require an examination are recorded in the test case report.

#### 4.5.5.6. Test Report

Test results are analyzed. The aim is an evaluation of the executed test and a decision on whether the test goal has been reached or whether the test has to be repeated. The outcome of each test is documented in a test report.

### 4.5.6. Resources for Quality Control

Through testing and reviews, the function of analytical QA is performed. This requires that the test process be controlled as part of the development process. Without suitable control mechanisms for the test process, the management does not have the necessary information to enable them to assess project progress and problems that arise. What are the quality control mechanisms?

#### 4.5.6.1. Tools and Aids

Few tools are specifically designed for quality control. These tools support quality control in all phases of software development. They are special tools for supporting SQA in error prevention, error recognition, error evaluation, and CC [Wallmüller 1994].

Tools for error prevention include documentation aids that provide partially automatic document writers, spelling checkers and thesaurus, data dictionaries, database generators, and syntax-driven editors.

Tools for error recognition and for error evaluation include static analyzers for complexity analysis (data flow analyzers, interface analyzers, and control · flow analyzers), software design tools, debugging aids, and file comparators, tools for checking standards, test case generators, and performance analyzers. Early warnings regarding possible execution time problems can be provided by simulators, execution time analyzers and performance monitors. Substantial software system testing can often be performed automatically by test suite generators and automatic test executors.

### 4.5.6.2. Metrics

Much attention has been devoted to questions associated with the measurement of quality. The quality assurance of products must be based on measurement. The earlier the measurement of quality begins, the earlier problems will be located [Fenton 1993].
The set of measurable values associated with the quality of a product is referred to as the product's quality *metrics.* Software quality metrics can be used to determine the extent to which a software product meets its requirements. The use of quality metrics increases the objectivity of the evaluation of product quality. Human evaluation of quality is subjective, and is therefore a possible source of disagreement, particularly between customer and developer.
A number of methods for establishing software quality metrics are currently being developed, though no generally accepted standard has yet emerged. For example, an initial draft of IEEE Std-1061 (1990) includes a detailed discussion of software quality metrics in general [Wallmüller 1994].
The basic approach for applying software quality metrics requires: the identification of all required software quality attributes which is usually derived from software requirement specification, the determination of measurable values to be associated with each quality attribute, a description of the method by which each measurable value will be measured, and a procedure for documenting the results of measuring the quality of the software product.

A set of many values can be used to determine the overall quality of a software product. However, a single measure can be created to represent the overall quality of the software product. This requires a weighted method for combining the measured quality attributes into a single measure of quality for the product.

Some examples of software metrics are:

-Reliability : the percentage of time that the system is successfully operational (23 out of 24 hours produces: 100* (23/24) percent).

-Recoverability: the amount of time it takes for the system to recover after failure ( 1 hour to reload from backups and 30 minutes to reinitialize the system).

-User-friendliness: the amount of training time needed for a new user.

-Maintainability: the percentage of corrections being right.

In the test process, measurable metrics exist. Test goals are considered an effective measurable metric which are used for assessing the scope and the results of a test. With reference to Schmitz, the following two test metrics can be defined. A test coverage metric (TCM) and a test result metric (TRM) [Wallmüller 1994].

Examples of TCM for the specification of test goals are:

$$\frac{\text{Number of executed branches}}{\text{Number of existing branches}}$$

The actual test goal for a module test might be 90% TCM.

As a test goal in module testing is the extent of test coverage. In system testing, many test goals are observed such as completeness, volume, load, security, and user-friendliness.

Examples of TRM are the mean time between errors found, or the necessary calculation time for a test.

The application of these metrics depends on the availability of tools for determining these factors.

So, the measurement of software quality is not be performed only at the end of a project. The degree of quality should be measured at regular intervals during development. Thus, any major reduction in the overall measure of quality should act as a warning for the PM that corrective action is required. High quality at the end of the project is achieved by assuring high quality throughout the development of the project.

#### 4.5.6.3. Quality Records

For the PM, it is essential to be constantly informed of the true status of the project. This is achieved by assuring the regular flow of objective and accurate information. Such information is acquired from reports by the QM.

Quality records are documentation which provide assurance that certain quality factors are present in the system. They include project status reports, project status meetings, test reports, and project reviews forms [Ince 1994].

##### 4.5.6.3.1. Project Status Reports

Status reports should be required from every member of the development team. The reports are submitted periodically, usually weekly or bi-weekly, and contain three sections [Bennatan 1992]:

-activities during the report period: each subsection within this section describes a major activity during the report period.

-planned activities for the next report period: each subsection within this section describes a major activity planned for the next report period.

-problems: each subsection within this section describes a major problem that either occurred during the report period, or that was reported previously and has not yet been resolved. This means that problems will be repeatedly reported until they are resolved. This section must explain why this report's section 1does not correspond to the previous report's section 2.

It contains also the date of report, the report period, the report name, and the name of the person submitting the report.

The preparation of a periodic status report should take about twenty minutes. Developers should submit their status reports to their team leader. The team leader then combines the reports of the team into a single status report, while maintaining the same report structure. This activity should take the team leader about thirty minutes.
The PM also receives status reports from other groups. The PM then prepares the project status report by combining the individual reports received into a single report. The project status report is then submitted to top management.
Project status reports are not submitted at the same frequency as internal project status reports. Project reports may be submitted bi-weekly or monthly.

#### 4.5.6.3.2. Project Status Meetings

In addition to the major life-cycle reviews discussed, project status meetings or reviews are conducted at intervals during the project life cycle. These reviews form the basis of any in-phase project reports as well as alerting the QM to any problems which may require an in-phase review, and providing background materials for milestone reviews.
Project status meetings are held periodically and regularly, for example once a week. It is attended by the QM, the PM and the development team representatives. The PM prepares for the status meeting by reviewing the status reports submitted, and scrutinizing the problem section.
The meeting begins with a report of project activities and general issues by the PM. Then each participant should be given about five to ten minutes to report on the activity of his area of responsibility. The discussion of problems should not be restricted to the person reporting the problem and the PM. All problems may be addressed to all participants, thus making their experience available throughout the

project. It is not the QM's role to provide solutions to the problems, but rather to guide the team members toward solutions.

Solutions should be worked out whenever possible during the status meeting. Any problem not resolved within five minutes should be postponed for discussion by the relevant parties after the status meeting [Bennatan 1992].

The proceedings of all project status meetings must be recorded. The record contains the date of meeting, name of meeting, present participants, absent participants, action items(name, action, date for completion), major decisions and items discussed.

Items discussed include project management changes and personnel changes, design status, development schedule status, coding status, software problem reports and corrections status, integration schedule status, testing status, progress on previous problems, and new action items/problems.

The record of the project status meeting is typed and distributed no later than by the end of the day. This is particularly important when there are action items to be completed on the same day.

#### 4.5.6.3.3. Review Control Forms

Two forms are proposed in controlling and documenting the formal SQA milestone reviews and audits.

Review preparation form

The QM with the review team-composed of development personnel review the project documentation using the appropriate SQA review checklists. The review is conducted using a review preparation form illustrated in Figure 4.3.

Review Preparation Form

Project: _____
Review Number: _____ Date: _____
Reviewers:_____
Pre-review meeting: [ ] Yes [ ] No

Review for testing of :
[ ] Requirement Definition [ ] Code [ ] Test Cases
[ ] Design [ ] Test Plan [ ] Other

Discovered Faults/Problems

| Fault Position | Description | Type | Corrective Action |
|---|---|---|---|
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |
| _____ | _____ | _____ | _____ |

Types of faults: interface (IF), logic (LO), input/output (I/O), documentation (DO), syntax (SY), test coverage (TC), standard (ST), other (O).

**Figure 4.3.** Review Preparation Form

Review Evaluation Form

After review completion, the QM prepares a review evaluation form illustrated in Figure 4.4. The QM discusses the review or audits results and record suggestions for any corrections or changes in the light of the findings, and schedule a second review, if applicable. When all changes and corrections have been made, the review team conducts a review using appropriate checklists. The QM reviews the review results and, if applicable, record any further suggestions for corrections or changes to be made in the light of the new findings.

At this point, the QM may approve the product as its is, or approve with the provision that recommended changes and corrections are implemented. These changes will be reviewed at the next scheduled life cycle review. Alternatively, the QM may refuse to close the review, instead calling for the changes and corrections to be implemented and another review scheduled.

At the formal close of the review, the QM prepare a report to top management summarizing the findings, results, and actions and all forms relating to the review, in the SQA project file.

Review Evaluation Form

Project:_____ Review Number:_____
Life Cycle Phase: _____
Checklist Number:_____

Comments/Analysis: _____
_____
_____

Review Phase Complete: [ ] Yes [ ] No

Next Review Date [if applicable]: _____

Signature: _____

**Figure 4.4.** Review Evaluation Form

## 4.6. Summary

QA is not a phase of the life cycle, it is an ongoing process to ensure that the product development process is being carried out according to the procedures laid down.
The quality system presented emphasized on two major disciplines: project management and quality management.
Following is a summary of the system.

| MAJOR REQUIREMENT | SUB-REQUIREMENT | DETAILED ACTIVITIES AFFECTING QUALITY |
|---|---|---|
| ***Management Responsibility*** | | |
| | *Quality Policy* | |
| | *Organization Structure* | |
| | *Management Review* | |
| ***QualityPlanning*** | | |
| | *Requirement Phase* | |
| | | 1. PDP (development methodology used, selection of tools, software documentation)<br>2. Test Plan (system test planning)<br>3. SRS Document<br>4. SRR<br>5. PDP Review |
| | *Design Phase* | |
| | | 1. Design Specification Document<br>2. DSR (PDR or CDR)<br>3. Updated PDP (coding standards, tools, techniques, code reviews, methodologies) + integration schedule<br>4. PDP Review<br>5. Updated Test Plan (component, integration, system), |
| | *Implementation Phase* | |
| | | 1. Updated Test Plan (Test cases+ procedures: component, integration, system, acceptance)<br>2. Unit Testing (code Review, test cases run)<br>3. PDP Review<br>4. User Documentation<br>5. Maintenance Plan |

| MAJOR REQUIREMENT | SUB-REQUIREMENT | DETAILED ACTIVITIES AFFECTING QUALITY |
|---|---|---|
| ***Quality Planning****(cont'd)* | | |
| | *Integration and Test Phase* | |
| | | 1. Integration Testing (module interface checking)<br>2. Product Testing (correctness, robustness, performance, documentation)<br>3. Acceptance Testing |
| | *Maintenance Phase* | |
| | | 1. Regression Testing (previous test cases run)<br>2. Documentation updates (due to corrective, adaptive or perfective maintenance) |
| ***Quality control*** | | |
| | *Static Tests* | |
| | | 1. SRR(unambiguities, completeness) use checklists and tools<br>2. PDR+CDR (performance, user interfaces, functionality, documentation)<br>3. POR after AT and PA (completeness of manual, documents, testing activities, monitoring of problems and corrections)<br>4. FA after PT (correctness of product functionality and its quality factors and the testing level, and documents)<br>5. PA after AT (correctness of AT run, documentation, proposition and implementation of solutions) |
| | *Dynamic Tests* | |
| | | 1. Test Planning (goals, methods, schedule for intended tests)<br>2. Test Designing (detailed instructions for test methods, test software and hardware specification, criteria used for test pass/fail) |

| MAJOR REQUIREMENT | SUB-REQUIREMENT | DETAILED ACTIVITIES AFFECTING QUALITY |
|---|---|---|
| *Quality Control(cont'd)* | | |
| | *Dynamic Tests(cont'd)* | |
| | | 3. Test Case and Procedure Planning (description of the execution of each test, input and expected output, and the actual execution of tests)<br>4. Test Execution and Reporting (analysis of results received from test execution, test pass/fail) |
| *Quality Control Resources* | *Tools and Aids* | |
| | | 1. Documentation aids, software design tools, debugging aids, checking standards tools, test case generators) |
| | *Metrics* | |
| | | 1. Identification of quality attributes<br>2. Associated values<br>3. Method for values measurement |
| | *Quality Records* | |
| | | 1. Project status reports<br>2. Project status meeting (status reports review)<br>3. Review control forms (review preparation form + review evaluation form) |

# Chapter 5

# Case Study

## 5.1. Introduction

This chapter provides a guidance for implementing the quality system proposed in the previous chapter. Although the quality system involves many elements, this case study emphasizes on and presents a sample of the QA activities that were conducted throughout the product development cycle in order to assure a quality product. It is intended to aid the SQA person and give him insight in performing his task.

The implementation was conducted at Mikati group, a Lebanese telecommunication company, in its billing department whose task is to develop billing software for the group. The case study concerns the CALLBACK billing software. The project is considered as an "in-house" software development where the customer requesting the development is taken to be part of the group. This service is provided by one of Mikati group Overseas company EASYDIAL.

## 5.2. Management Responsibility

Two requirements are necessary: Quality policy definition and the organizational structure.

### 5.2.1. Quality Policy

Management's first step in producing quality products is to define and document its quality policy. The quality policy at Mikati group is defined as follows:

Mikati group

The revenues and the profit of every company in this group depend on the quality of the products which we produce. This group is committed to meeting our customer's and subscriber's expectations in terms of the quality of the products and the services which we provide. Our customers and subscribers expect faultless products and services. Our intention is to achieve this by preventive measures and the application of an effective software quality assurance program.

We recognize that quality is everyone's responsibility. The entire team must adhere to the company's quality policy and are obligated to make it a success.

Signed by _____

General Manager

Date _____

### 5.2.2. Organizational Structure

Since the company decided to introduce a quality program, then a person responsible for this program is designated. In this project, only one person can fulfill this task. het is called the SQA person, or QM. He has a reporting channel to senior management, the GM, that is independent of the PM. The internal independence of the QM is

necessary so he/she can perform his/her job without being influenced by project schedule and cost pressures.

The QM has a job description statement, as all employees in the company, stating his responsibilities. This is illustrated in Figure 5.1.

Job Description

Job Title: Quality Manager
Department: Computer department
Reports to : General Manager
Function:
To provide insight in the introduction of a quality program, control the development process, and provide assurance that the product satisfies customer requirements.

Responsibilities:
1- Participation in the development of plans, such as the project development plan.
2- Approval of the development methodology and tools.
3- Planning and supervising and approval of testing.
4- Ensure that adequate reviews are carried out.
5- Ensure that appropriate software documentation are present.
6- Conduct audits to ensure that the developed product is in accordance with the organization policy.
7- Administration of configuration control.
8- Provide management with confidence that objective information on the process and products of the project is being reported.

**Figure 5.1** QM Job Description

In every organization, anybody having anything to do with the development of the software, contributes to the quality. All responsibilities and authorities are clearly established and understood. Along with his responsibilities, the QM has the authority to approve or reject product components, and to give permission for the project to proceed from its current phase to the following phase.

Working relationships between all personnel who manage, perform and verify work affecting quality are best represented and clearly defined in the company organizational structure.

The organizational structure at the company is illustrated in Figure 5.2.

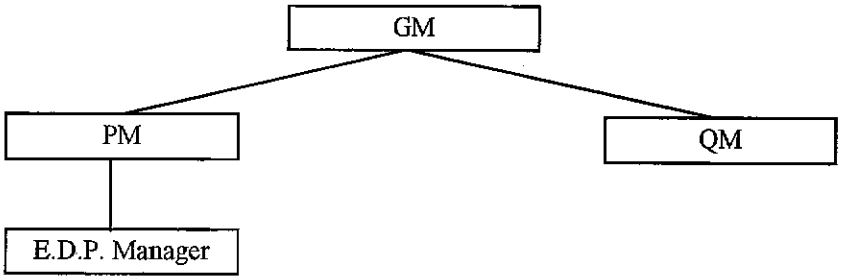


**Figure 5.2** Company Organizational Structure

The *GM* is responsible to the president. He is also responsible for setting the company's quality policy and establishing new organizations, and thus notification of new projects.

The *PM* directs, controls, administers and regulates a project to build a hardware/software system. He is responsible to the customer, which is, in this case, another organization in the company, and he is responsible for preparing the project plan. A reporting relationship exists. The E.D.P. Manager reports to the PM. Although, he sometimes reports to the GM. As to the PM, he reports to the GM.

The *E.D.P. Manager* has responsibility of performing the software development and maintenance activities (requirements analysis, design, code, and test) for a project.

The *QM* has the responsibility of planning and implementing the project's QA activities, including testing, to ensure the software process steps are followed and quality products, satisfying their requirements, are developed. He reports to the GM, and he works closely with the PM and the E.D.P. Manager.

## 5.3. Quality Planning

In order to meet quality requirements, many activities are required of the PM and the QM at the commencement of the project and throughout the project life-cycle. The management of each phase is weighted equally to achieve a final quality product. The QA activities performed throughout each phase and at the end of each phase are presented. As to the PM's activities, they are discussed briefly, making references to standards when necessary, since the emphasis is on the QA activities.

As mentioned in the previous chapter, the second element of the quality program: quality planning, is performed in two phases: initial planning and final planning.

## 5.4. Initial Quality Planning Phase

Normally, in this phase, the proposal phase, the objective is to win contracts. How the tasks will be performed and what the end product must do are established in a legal and contractual sense. The contract is reviewed by the company developing the product to ensure that [Bennatan 1992]:

1- the requirements are adequately defined and documented, and quality requirements are established.
2- any requirements differing form those in the offer are resolved, and all deliverables are identified.
3- it has the capability to meet contractual requirements, that the product can be delivered with all its requirements satisfied within the time-scales specified by the customer and to a specified cost.

The developing company should be confident that the software can be delivered within the constraints specified by the customer. These constraints include system-specific

constraints such as response time, memory occupancy limits, file occupancy limits, compatibility with other software or hardware, and the satisfaction of a set of functional requirements. They also include specific project constraints such as required duration and cost.

Although the proposal phase is not considered a formal phase, all the above is necessary and applies when a project is contracted out, because in this case, this phase provides the basis for the definition of the software requirements and initial planning and preparation of estimates. At such an early phase, there is a need to calculate a rough cost for the project. Standards exist which enable the tasks in the project to be identified and approximate costs given. These tasks are at a high level, and they expand if the project proceeds [Bennatan 1992].

When the supplier is awarded a contract for a software project, a formal contract is issued and signed by all parties. Then, all aspects of the contract are reviewed- both legal and technical.

In our case study, no contract is involved since the computer department is producing the billing software for one of the group's company. As a result, the life-cycle of the project begins at the requirement phase which is the first formal development phase in the final planning phase, proposed in our quality program.

It is important to point out that there must be some form of internal agreement or contract between the computer department and the company requesting the software because the software department should behave as professionally as an external software company.

## 5.5. Final Quality Planning Phase

This phase includes all the mandatory phases of software development. In this phase, what are the SQA activities conducted by the QM and how they are conducted is described. The management issues affecting quality associated with each phase are also described.

### 5.5.1. The Software Requirements Phase

The life-cycle begins with the software requirements phase. This phase provides a detailed description of the billing software system to be developed. This description is laid out in the SRS document according to which the software is tested at the end of the project to demonstrate that the required product has indeed been produced.

EASYDIAL company designated a representative- one of its team members for dealing with the E.D.P. department and clarifying any ambiguous issues. His tasks include:

1- defining the software requirements to the developer.
2- answering questions from the developer.
3- approving developer's proposals.
4- defining acceptance criteria.

This phase produced two documents: one main product document that is the Software Requirements Document (SRD), and one project planning document, the PDP. Although a test plan might be produced at this stage, however, system test design cannot really begin until the first stage of the implementation phase. At this point there will be sufficient product detail available to allow the drafting of the test plan. So, no test plan is produced at this stage.

### 5.5.1.1. SRD Production and Review

The standard used to produce the SRD is that proposed for the Lebanese market [Hajjar 1996]. The SRD is given in Appendix A.

After establishing the system requirements, the QM performed checks as to their adequacy. He conducted reviews for checking the process of establishing these requirements, change methodologies, or enforce the procedures set by the standard. Any vague or suspect requirements were deleted. The reviews conducted were chaired by the QM, in the presence of the E.D.P. Manager and EASDYDIAL's representative. The aim of the review was to ensure that the requirements are correct. The reviewers went through the specification document, ensuring that there are no misunderstandings as to what is meant by each statement in the document, using a checklist and a review preparation form.

Table 5-1 illustrates a sample checklist that was used in the SRR.

| Software Requirement Review Checklist |
|---|
| Project: _____<br>Checklist Number: _____ |
| 1- Are all requirements included?<br>2- Is each requirement clear and does it have the same interpretation to all who read it?<br>3- Are all requirements documented, assuring that no verbal understanding remain?<br>4- Can we prove that each requirement has been met?<br>5- Does any requirement conflict with any other requirement?<br>6- Is the requirements specification documented in a way that enables it to be easily corrected or changed later?<br>7- Are all necessary hardware resources specified?<br>8- Have all the functions that the user needs been identified and specified?<br>9- Has the acceptable level for accuracy been specified for results?<br>10- Are the requirements comprehensible for those who have to design the project? |

**Table 5-1** SRR Sample Checklist

The requirements has evolved gradually. Some of the requirements were found ambiguous during the review, others were missing. As an example of an ambiguous requirement is that stating that reports are generated. This statement was updated to specify the types of reports needed which are important for controlling the subscriber calls and providing the status of the company. After review completion, the QM prepared a review evaluation form. The review was discussed and suggestions recorded on the review preparation form, for the corrections and changes in the light of the findings, and the QM scheduled a second review. When all changes and corrections have been made, the review team conducted the second review using the same checklist. The results were reviewed and the QM approved the document. So, two versions of the specifications were produced until convergence to the final approved document. At this point, the second and final review declared the requirement document as the first approved project baseline. It was approved by the QM.

#### 5.5.1.2. PDP Production and Review

The standard used to produce the PDP is that suggested in [ISO 9000-3 1991]. This plan was developed by the PM. It identifies the resources and schedule required to develop the billing software. it shows how the phases are implemented and identifies:

- the inputs and outputs to each phase
- schedule and resources for each phase
- progress control and status methods
- tools and methods to be used
- verification procedures for each phase (reviews, audits and testing)

The PDP produced for this project is given in Appendix B. It is essential to note that it is not a detailed PDP and it serves only as a sample on which the QM will work on and

will have to review. The aim is to state that a PDP must be produced by the PM to be reviewed by the QM.

After the PDP has been produced, the QM checked every aspect of the plan. He conducted a review in the presence of the PM. The aim of the review is to check the most important aspect of the plan, which is the project schedule.

A sample of the checklist that was used by the QM during the PDP review is presented in Table 5-2.

The aim of the review is to convey the status of the project as compared with the plan. It is very crucial to the project management because when there is a failure to status the development effort against the PDP, then the organization loses control and cannot make the necessary changes in the engineering process to deliver a product that meets requirements within schedule. The changes include adjustment of resources and schedule. Management, through reviews, is aware of any influences that affect the project's budget and schedule or the quality of the product.

The PDP approved by the QM, the permission for the project to proceed was given. The next phase is the design phase.

### 5.5.2. The Design Phase

The design phase is the second phase in the software development cycle. This phase is split into two parts: system or architectural design and detailed design.

During this phase, an architectural, detailed design, user interface, database design and error handling design are created that describe the software solution for the requirements specified in the SRS. In addition, in the second stage of this phase, the detailed design phase, an initial version of the test plan is drawn.

As a result, the outputs of this phase are the SDD and a test plan, and this phase is completed when the following have been reviewed and received approval: SDD and test plan, along with a review of the PDP.

While the system design is concerned with the development and specification of an architecture for the modules and the development of data architecture which specifies how the data is to be implemented in terms of variables, files and tables, the detailed design is the process of filling out details within the system design in terms of a program design language. However, the detailed design may be skipped and only the system design can be carried out. Thus, preferring to transform the system design into program code directly, for small projects [ISO 9000-3 1991].

In the sample documents that are presented: the SDD and test plan, only the major function of the billing software: the billing processing function, presented in the SRS document, is designed and tested.

#### 5.5.2.1. SDD Production and Review

The design of the product dictates to a great degree its quality. The activities in the design phase must be carried out in a disciplined manner, in order to produce a product according to specifications rather than depending on the test and validation activities for assurance of quality.

The standard used to produce this document are the guidelines proposed by ISO9000-3 [ISO 9000-3 1991] given in Appendix C.
This phase cannot conclude until a review of the design document is conducted to set the second major baseline. The review was held and chaired by the QM. A sample checklist used is presented in Table 5-3.

| Software Design Document Review Checklist |
|---|
| Project: _____<br>Checklist Number: _____ |
| 1- Has the standard chosen not adhered to ?<br>2- Is the system identifiable as modules and are these listed?<br>3- Is the software specification complete, consistent, and unambiguous?<br>4- Is there traceability of the requirements through the specifications?<br>5- Has user input been kept to a minimum?<br>6- Are there any indication of non-fulfillment of performance requirements?<br>7- Are user interfaces uniform?<br>8- Are the screen outputs clear?<br>9- Are the screen layouts not overloaded with information?<br>10- Are there missing or unused variables in a module?<br>11- Is the design description incomplete, ambiguous?<br>12- Are the algorithms in a module clearly specified?<br>13- Is the system designed in such a way that it can be progressively built up and tested?<br>14- Are the modules developed in such a way that they are testable, maintainable and usable? |

**Table 5-3** SDD Review Sample Checklist

During the review, discrepancies were recorded on the review preparation form. Corrective actions taken, and the QM approved the SDD. Thus, the second major baseline was set.

However, in this phase, an initial version of the test plans are drawn for different tests. The QM participates in the development of such plans. He supervises and approves their development.

### 5.5.2.2. Test Plans, Test Design Development and Reviews

A test plan is developed at this stage to describe in detail the tests required to ensure that the product has met its functional requirements, and performs in the manner expected.

These plans are developed for each level of testing. They correspond to the developed product of the phase. Normally, the requirement phase produces system and acceptance test plans. However, since no sufficient data is available to allow the production of the test plans, the test plans are produced in this phase-the design phase for each of the following tests: component, integration, system and acceptance.

Certainly, since we are dealing with a small project, the test activities are narrowed down and can be combined to minimize the time frame required for their fulfillment. Instead of having five activities, they are reduced to three activities:

- Test plan and test design creation
- Test cases and test procedures creation
- Test execution

The reason behind this suggestion is that the test design refines the test plan approach, identifies specific features to be tested by the design, and define its associated test cases and procedures. To eliminate the duplication of information, we have combined them.

For our project, a test plan is developed for testing each component of the billing system, and taking into consideration the test design. As a result, instead of having two documents for the same test, we generate one document: the test plan with the detailed information relative to the design. Similar test plans are prepared for integration and system testing. No acceptance testing is considered necessary, since the data used during testing is real CDR's. Once the monthly invoices are issued and they are correct, the system is accepted. At this point, we can notice the change in the schedule set in the PDP.

However, the activities in the schedule are of importance in the case of large projects and they must be taken into consideration.

For the purpose of this study, the focus is on the reviews of the test plans. So, no test plans are presented.

To ensure their correctness and completeness, test plans need to be reviewed by the QM.

A sample of the test plan checklist used by the QM is presented in Table 5-4, taking into consideration the test planning and test design.

| Test Plan and Design Review Checklist |
|---|
| Project: _____<br>Checklist Number: _____ |
| 1- Has the standard chosen not adhered to ?<br>2- Has a schedule of the testing activity been derived?<br>3- Has the tasks to be performed been specified?<br>4- Have the people responsible for the testing activities been designated?<br>5- Has the test environment been specified?<br>6- Have the inputs and outputs of such a plan been specified?<br>7- Have the features to be tested been specified?<br>8- Have the features not to be tested been specified?<br>9- Have the entry and exit criteria for each phase of testing been specified?<br>10- Is any testing tool required?<br>11- Is the test plan in parallel with the design?<br>12- Have test goals been set?<br>13- Have test cases been defined to reach the test goals?<br>14- Have test methods been determined to derive test cases?<br>15- Has an estimate of the number of test cases and their duration been derived? |

**Table 5-4** Test Plan and Design Review Sample Checklist

Test plans developed and reviewed and approved by the QM, the next phase is the implementation.

### 5.5.3. The Implementation Phase

During this phase, the software design is implemented and the various modules that make up the software are tested at the unit level and the functional level. The purpose is to implement the design and prove that the modules created fulfill the functionality of their design. They are integrated later into a system. In addition, test cases and procedures are generated for component, integration and system testing.

With the participation of the QM, test cases and procedures are created for the component, integration, and system tests, whose plans and designs were created in the previous phase.

The test cases specify in detail for each test case executed on a specific object, the input, and the expected output. In addition, the test procedure describes how to actually run the test.

While generating the test cases and procedures, the QM is certain that three key points are not missing:

1- the inputs and outputs are detailed for each test case

2- a test procedure is specified for each test case

3- the expected results are specified for each test case

The third activity, performed by the developer, after coding and test case and procedure generation, is the actual execution of module testing. After the test is run, the expected results are compared to the actual results determining whether the module tested has passed the test. While conducting these tests on the modules of the billing software, bugs were detected and fixed.

As coding progresses, the integration phase begins to overlap with the implementation phase. The modules that are tested are released in the order required for integration. Then, integration testing is executed according to the procedures set. When the integration testing is complete for all the software modules pieced together, then the system is ready for the system testing and the testing phase begins.

### 5.5.4. The Testing Phase

Normally, this phase represents the final testing process and is split into two activities: system and acceptance testing. However, as mentioned earlier, since the billing system testing is run on real data, then system testing can be considered as acceptance testing. During this phase, the billing software is tested to ensure that call records are processed correctly, subscribers charged according to the set billing process, and invoices issued correctly. Errors were found. They were fixed by the developers, and

the CDR reprocessed. This phase is complete when all the requirements are met. Finally, a test report is generated by the QM providing management with the reliable, needed information confirming the readiness of the software for release.

During system testing, the QM using a checklist, ensures that some key points are tested. Table 5-5 presents a sample system testing checklist.

| System Testing Sample Checklist |
|---|
| Project: _____<br>Checklist Number: _____ |
| 1- Does the system meet all its functional and non-functional requirements?<br>2- Was the system subject to volume testing?<br>3- Was the system subject to load testing?<br>4- Was the user-friendliness of the system tested?<br>5- Was the security of the system tested?<br>6- Was the system efficiency tested?<br>7- Was the user documentation tested for accuracy? |

**Table 5-5** System Testing Sample Checklist

After the management approval of the software system, the billing software is ready to be delivered to the company requesting it, and the maintenance phase begins.

### 5.5.5. The Maintenance Phase

This phase is considered as part of the CM. However, we concentrate on the reviews and audits that are conducted after product release to the customer.

Normally, in this phase, the QM conducts audits. After the system testing is run, and after product release, the QM conduct the FA. It is primarily concerned with the actual functioning of the software product. Its purpose is to ensure that [Vincent 1988]:

1- all required fixes have been, or are being, implemented

2- the testing required was properly conducted

3- the test results indicate that the software product meets performance specifications and requirements, including those required software quality factors
4- all test documentation is properly prepared
5- all problems, deficiencies discovered during testing have adequate solutions proposed and implemented

Upon approval by the QM, the third baseline-product baseline is established.

After acceptance test is run, the PA is conducted. The primary intent of the PA is to [Vincent 1988]:

1- verify that the product is in compliance with the organization standards and policy
2- ensure that the product reflects required user, and quality standards
3- ensure that the software and its documentation are internally consistent
4- ensure that all problems, deficiencies discovered during testing have adequate solutions proposed and implemented
5- ensure that all modifications made to the product are in accordance with the specifications and requirements settled in the product baseline to ensure that all system manuals are in good order, and in conformance with the final product configuration
6- ensure that all support programs and hardware are in good order, and in conformance with the final product configuration
7- ensure that the software product is ready for delivery
8- ensure that all implementation plans are in good order, and are acceptable to the user, the quality and project management

Once the QM has given its approval, the software product will be released to the user for initial operation.

At this point, the POR is accomplished during the operational and maintenance phase, after the PA, and not sooner than ninety, nor later than one hundred and eighty, days after the delivery of the software product to the user.

Prior to the POR, the system designers and development team members place the software system in operation, and the test team performs any supplemental site tests.

During the POR, the system designers review and interpret the results of the supplemental site tests, and explain any problems encountered in the testing and the way to solve them.

Among the primary objectives of the POR are to [Vincent 1988]:

1- verify the adequacy and completeness of system support manuals
2- verify the adequacy of management plan, and its proper implementation and use
3- ensure that all supplemental site testing is carried out completely and accurately
4- verify that all test documentation is properly prepared
5- ensure that all problems, deficiencies discovered during testing have adequate solutions proposed and implemented
6- ensure the continued quality of the software product through continued quality review, and the monitoring of problems and corrections
7- ensure the maintainability of the product through a monitoring of design changes

Upon approval by the QM, the operational baseline is established. This baseline is set at the conclusion of the development cycle and finalizes the development of the software product.

None of the above mentioned audits, and review were conducted on our project. The project was completed and entered its maintenance phase when the QM approved the system testing. The requirements were met. The billing system billed the subscribers correctly with an efficient processing of the calls, and a security system in place.

This case can be considered as a special case because the testing was conducted directly on real data, second the relationship between the organization developing the product and the organization requesting it does not require such audits and review to be conducted after the product is released. So, normally, the management of the organization developing the product must decide-based on these factors whether to conduct such audits and review after product release.

# Chapter 6

# Conclusions

Software producers are expected to produce high quality products on time and within budget. In Lebanon, we are way far from reaching this goal since the complexity of software is still underestimated by management and developers, who try to improve quality by testing instead of developing quality step by step, and development is conducted in the absence of a systematic development process.

So, the quality of software is facing a problem and it stems from the quality of the development process. By focusing and improving the quality of the process, we improve the quality of the software product.

Based on a study conducted by Hajjar in Lebanon it was found that [Hajjar 1996]:
the software engineering practices do not use any standard methods, the software development phases are not clearly subdivided, no actual plans are used or updated, methodologies in requirement analysis fail to present clear and proper requirements and thus quality criteria, testing is misinterpreted since it is regarded as a method to prove that there aren't any errors rather than a method to find errors, and no measures are used to assess the quality of the software development.

In other words, unfortunately, the development process in Lebanon is applied based on subjective assessments and self-made rules, details methodologies, documentation and planning issues are not the main concern, but rather the software as a whole. Add to this, that assuring quality in software is regarded as a burden that can delay the product from emerging and increasing its cost. Whenever, the software works, free of any

errors and whenever the customer is satisfied, that is field proof that the software is of good quality.

Certainly, this conception is incorrect and must be changed. This has lead to the need of the Lebanese software firms for a quality system to be introduced into their companies, so that planning and production is carried out systematically to help produce an acceptable software on time and within budget.

Currently, in Lebanon, we cannot jump a big leap in to practicing and implementing a set of international standards, such as ISO and IEEE collection on software engineering. But, we can certainly try to initiate our own quality system that meets the Lebanese market and that is accepted by companies' managers. This was the aim of our research.

This thesis has represented a quality system for small-to-medium Lebanese firms in order to attain an acceptable level of quality if not an international level in the software production industry. It contained the minimum acceptable requirements for the introduction of quality system and has focused on the minimum activities to be carried out in each of the development phases. Like ISO series of standards, this quality system is based on the premise that if the production is right, the product produced will be right. The aim was to produce a *simple* and *direct* quality system integrated into the entire development process and accepted by all Lebanese firms' managers. We have tried to show how things are done through the case study presented. The system was implemented in three months, and the company noticed the importance of the quality system. It has adopted and carried out most of the activities required by the system, which are applicable at the company.

Software engineers in Lebanon should realize the importance of SQA. Quality assurance must be looked upon as a long term investment. While it will result in

extension of the project cycle, it will save time and resources during the operation and maintenance phases. They must be aware that the achievement of quality is the responsibility of every person involved in the delivery of the software product. So, everyone has a role to fill it and the extent to which they fill it determines and leads to better software product.

The suggestions presented in this thesis present a starting point to initiate a quality system adapted to the Lebanese firms. However, based on our experience in implementing this system we found out that some refinements are needed. Such refinements include adapting selected development standards to the Lebanese market, the integration of CM and a system of metrics within the quality system which were not considered for the purpose of the thesis, testing activities, reviews and audits were minimized, documentation-one of the most important element for the success of the quality system, and maintenance were not discussed in detail.

Finally, I would like to conclude this thesis by a fact that quality is never by accident, it is always the result of an intelligent effort.

# Appendix A

# SRD for the Callback Billing System

## 1. Introduction

The introduction includes an identification and an overview sections.

### 1.1. Identification and Scope

This is the Software Requirement Specification document, where the task to be performed by the software are refined into some form of requirement analysis. The software requirements produced are clear and explain what the software is expected to do.

### 1.2. Overview

A billing system is required for the callback service offered at EASYDIAL for their international access calls. The switch providing the service is located in New York, and the billing is based on the 2-leg billing process: (leg a) and (leg b) where (leg a) is the link from the switch to the country where the subscriber is, and (leg b) is the link from the switch to the country called by the subscriber.

## 2. Program Set Descriptions and Objectives

The system is billing the subscribers and thus must produce an invoice for each one of them, along with a summary invoice and a detailed invoice and to provide necessary reports after billing process in order to get the company's status and profit margin

## 3. Functional Requirements

Functionality's three components are:

### 3.1. Input Requirements

- One file forms the basis and the starting point to the billing process. This file is the Call Detail Record (CDR) billing file which collects and records the subscriber identification and usage time on calls made through the switch. This file is used for subscriber billing, and is made available to the billing department for processing as agreed with the EASYDIAL's representative (once a week or once a month).

- The CDR billing file contains CDR's, each of which contains calling and terminating numbers, duration of long distance calls for billing use.

A CDR occurs for each call attempt, that is whenever a subscriber calls the switch, and waits for a callback, and then gets instructions about how to access the system and starts dialing.

- A CDR is 165 bytes long and is formatted in ASCII. Table A-1 illustrates CDR record format with a description of each field.

| Field # | CDR | Field Description | Length | Type |
|---|---|---|---|---|
| 1 | CDR type | (C=calling card, I=Callback) | 1 | Char |
| 2 | In date | Date call comes into switch | 6 | Num |
| 3 | In time | Time call comes into switch | 6 | Num |
| 4 | Answer date | Date (leg b) is answered | 6 | Num |
| 5 | Answer time | Time (leg b) is answered | 6 | Num |
| 6 | Dial date | Date (leg b) dialing begins | 6 | Num |
| 7 | Dial time | Time (leg b) dialing begins | 6 | Num |
| 8 | Outdisc date | Date (leg b) disconnected | 6 | Num |
| 9 | Outdisc time | Time (leg b) disconnected | 6 | Num |
| 10 | Indisc date | Date (leg a) disconnected | 6 | Num |
| 11 | Indisc time | Time (leg a) disconnected | 6 | Num |
| 12 | Calldisc date | Date call disconnected from switch | 6 | Num |
| 13 | Calldisc time | Time call disconnected from switch | 6 | Num |
| 14 | Acct code | Account code used for call | 10 | Num |
| 15 | CDR ID | CDR identifier DID(callback) PIN(calling card) | 10 | Num |
| 16 | Destnum | Destination (leg b) number dialed | 20 | Num |
| 17 | Ani | Ani of caller or callback number | 20 | Num |
| 18 | International | Was this an international call (Y/N) | 1 | Char |
| 19 | Dnis | DNIS for call | 20 | Num |
| 20 | Complete | Was call answered (Y/N) | 1 | Char |
| 21 | Duration | Duration of call in seconds | 10 | Num |

**Table A-1** CDR Description Format

- Each CDR is considered either a successful call or an unsuccessful call (attempted call) depending on field #20.

- The billing is based on the 2-leg billing process: (leg a) and (leg b)
  where :
  (leg b) is defined as the duration in field # 21 of a call. It is zero in the case of an attempt.
  (leg a) is defined as the duration of (leg b) + a time delay equals to the time a customer takes to listen to the recorder before start dialing. (leg a) processes in parallel with (leg b), and can be derived from field #13 and field #3 as their difference [field #13 - field#3].

However, the value of (leg a) changes and depends on whether the call was the first one made and whether it was an attempt or not.

- Successful calls charging

In the same session, a subscriber can make consecutive calls. That is, when the subscriber calls the service and he is called back and is given a dial tone, he can make one or several calls in the same session. These calls will be either successful or attempts.

However, the subscriber is given **ONLY** 90 seconds free on the **FIRST** call placed, if (leg a) duration is greater than the duration given as free- 90 seconds. If this is the case, then the final value of (leg a) is (leg a) = (leg a) - 90. Otherwise, (leg a) value remain unchanged and it is less than or equal to 90.

Normally, the final value of (leg a) is greater than the value of (leg b). If this was not the case, then the final value of (leg a) is set to the value of (leg b).

- Attempted calls charging

In the case of an attempt, (leg b) is zero since the subscriber may have not answered the callback call, or after dialing completion, a busy tone was given or the called party did not answer. In this case, the subscriber is charged only (leg a) duration which is derived as follows.

The subscriber is given 60 seconds free **ONLY** if the current call under process is the **FIRST** one in the session. If this is the case and (leg a) is greater than 60, then (leg a) is computed as (leg a) = (leg a) - 60. Otherwise, (leg a) is set to zero.

- (leg a) and (leg b) tariff determination

After derivation of the final (leg a) duration and (leg b), the tariff corresponding to each one of these values must be determined. The tariffs are derived from two main rate tables. One rate table corresponding to (leg a) and another one corresponding to (leg b). Both tables are determined as New York origin because the switch location is New York.

(leg a) tariff is determined as the tariff from New York to the called back country-where the subscriber is, and (leg b) tariff is determined as the tariff from New York to the called country.

- the Formula applied to calculate a call charge becomes:

(leg a) * (rate a) + (leg b) * (rate b)

3.2. Performance Requirements

This requirement include static requirements and dynamic requirements.

3.2.1. Static requirements

The billing software is designed to support one user or multi-user environment. As to the number of files handled, in our case one CDR file is generated for each day. These files are processed so that all the CDR's are gathered in one file for later processing. Several files are also needed.

-subscriber file

-parameter file

-itemize file

-invoice file

-history file

3.2.2. Dynamic requirements

The number of records to be processed each month is approximately 75000 CDR, varying depending on the increasing number of subscribers. This is a variable number and depends on many factors. Some of the factors is the number of calls made by each subscriber and thus the performance of the system.

3.3. Output Requirements

As a result of processing the CDR billing files, the output produced is one file whose records are processed on subscriber basis. Each subscriber is charged for the calls he made during the month. At the end of the month, bills are produced for each subscriber whose charge is greater than zero. Two types of invoices are produced: a summary invoice, and a detailed or itemized invoice.

After processing each CDR file, a history file is necessary to record the first and the last records in each processed file. It is necessary to prevent duplication and the processing of the same file more than once.

## 4. Non-Functional Requirements

Several characteristics are determined for each project. The characteristics that are important for this project are correctness, reliability, efficiency, and integrity. Design constraints are also considered as non-functional requirements.

4.1. Correctness Requirement

The software has to be correct and must fulfill its objectives: to bill the subscriber correctly and produce correct invoices.

4.2. Reliability Requirement

The software is not expected to fail. However, there are some cases where failure occurs. For example, in the case of a change in the CDR format or some error in the data file such as an uncompleted call record.

4.3. Efficiency Requirement

This requirement falls into two categories: execution efficiency and storage efficiency. It is concerned with the use of resources such as the processor time, and storage capacity.
The software has a better performance if it is run on a high-speed processor and has a large storage capacity since the data that need to be processed is large.

4.4. Integrity Requirement

There are some functions in the software that must not be accessed by all users. These functions need to be protected from unauthorized access. These include files creation, rate table additions and updates, and tariff reports production.

4.5. Design Constraints

Three design types are considered: software design constraints, hardware design constraints and user design constraints.

The language used to develop the software is Micro Focus COBOL version 3.2 under SCO UNIX version 3.1 on an Intel Pentium platform. The software operates only under such environment.

As to the hardware design constraints, no constraints are imposed on the hardware under which the software is to be developed.

As to the user design constraints, no constraints are imposed on the user of the system.

## 5. Approvals

| | |
|---|---|
| Project Manager | Date |
| E.D.P. Manager | Date |
| Quality Manager | Date |

# Appendix B

# Project Development Plan

## 1. Introduction

This section includes an overview of the project and its deliverables.

### 1.1. System Overview

The project to be developed is a billing software for the callback service. The callback service is a system through which the subscriber will be called back by a vocal server located in the U.S. providing him with a U.S. dial tone. This tone can be used to call internationally while being charged according to specific rates applied in this area toward other international destinations. The billing software charges the subscriber for the calls he makes.

### 1.2. Project Deliverables

The billing software is the only item to be delivered to EASYDIAL company.

## 2. Software Development Management

This section specifies the project organizational structure, and defines responsibilities for the various project elements.
In our case, the organizational structure is the same as the one presented in section 5.2. of this chapter. So to avoid repetition, they are not presented in this section of the plan.

Normally, they must be stated also in this section: the project organization and resources used to develop the product, and the organizations that are used to support the development effort, along with their responsibilities.

## 3. Schedule and Milestones

This section includes the scheduled activities, milestones and baselines, and budget administration. It also involves a schedule of planned major QA activities in relation to project milestones.

### 3.1. Scheduled Activities

Many ways to represent the schedule: list of activities, diagrams, or graphs. The most common methods of schedule representation are PERT network diagrams, GANTT charts and list of milestones. The method used to schedule this project is a scheduled activity list. The schedule is intended to provide information related to the activities and time of implementation. It is useful for assigning activities (low-level tasks) to project personnel.

Table B-1 contains the project schedule list of activities.

| Activity ID | Activity Name | Description | Start Date | End Date | Dependency | By |
|---|---|---|---|---|---|---|
| 1 | Requirement | System requirement | | | | |
| 1.1 | Equipment | Equipment procurement | 5-Jan | 20-Jan | 1.4 | PM |
| 1.2 | Staffing | Assign team members | 5-Jan | 20-Jan | 1.4 | PM |
| 1.3 | Estimates | Obtaining an initial budget to fund basic development equipment | 5-Jan | 25-Jan | 1.4 | PM |
| 1.4 | PDP | Prepare the project plan | 5-Jan | 25-Jan | | PM |
| 1.5 | Review PDP | Updated PDP | 26-Jan | 31-Jan | 1.4 | Review team |
| 1.6 | Review SRS | | 26-Jan | 31-Jan | | Review team |
| 2 | Design | System design | | | | |
| 2.1 | Test plan | Prepare test plan(component, integration, system, acceptance) | 1-Feb | 25-Feb | | Test team |
| 2.2 | Test design | Prepare test design(component, integration, system, acceptance) | 1-Feb | 25-Feb | 2.1 | Test team |
| 2.2 | Review PDP | Updated PDP | 26-Feb | 28-Feb | | Review team |
| 2.3 | Review SDD | | 26-Feb | 28-Feb | | Review team |

**Table B-1** Sample Activity List

| Activity ID | Activity Name | Description | Start Date | End Date | Depend on | By |
|---|---|---|---|---|---|---|
| 3 | Implementation | System coding | | | | |
| 3.1 | Software coding | Development of code | 8-Feb | 15-Mar | | E.D.P. dept |
| 3.2 | Test cases generation | Prepare test cases(component, integration, system, acceptance) | 8-Feb | 15-Mar | | Test team |
| 3.3 | Test procedures generation | Prepare test procedures(component, integration, system) | 8-Feb | 15-Mar | 3.2 | Test team |
| 3.4 | Test execution | Component test execution | 15-Feb | 28-Feb | 3.2,3.3 | Test team |
| 3.5 | PDP Review | Updated PDP | 10-Mar | 15-Mar | | Review team |
| 4 | Integration and testing | System integration and testing | | | | |
| 4.1 | Module integration | Construction of software system from various components | 15-Feb | 28-Feb | | E.D.P. dept |
| 4.2 | Integration testing | Integration test execution | 20-Feb | 15-Mar | 4.1 | Test team |
| 4.3 | Test procedure | Prepare test procedure for acceptance testing | 15-Mar | 30-Mar | | Test team |
| 4.4 | Product testing | Full functional system-system test execution | 15-Mar | 30-Mar | | Test team |
| 4.5 | Acceptance testing | Live system run-acceptance test execution | 30-Mar | 30-Apr | 4.3 | Test team |
| 4.6 | Test report | Prepare test reports | 30-Apr | 5-May | | QM |

**Table B-1** *(Cont'd)* Sample Activity List

This plan is required at the initial stages of the project. Unfortunately, a full list of activities is usually not available until well into the design phase. Therefore an initial version of the project schedule usually starts with a list of high level activities, and this initial schedule is repeatedly refined as more information becomes available.

3.2. Milestones and Baselines

Certainly, not all activities are of equal importance. Some activities signify major events in the project development cycle. The completion of the requirement specification is a major milestone, as is the completion of the software design specification.

The most important project event is the conclusion of the project, signified by the successful completion of the AT. These important events needs special attention and are recorded in a separate list of major project milestones. Normally, milestones are used as points of payments. However, since our project is considered to be an in-house project, the milestones are used for the measurement progress on the project and for determining baselines.

Since milestones are described as major events, then baselines are described as major milestones. Baselines refer to critical points during software development where major decisions are finalized. Project reviews are where such decisions are finalized. Three major baselines:

- The Functional baseline which is set at the end of the requirement phase-at the system requirement review-to finalize the system functional requirements.
- The Allocated baseline which is set at the end of the design phase-at the design review.
- The Product baseline which is set at the conclusion of the development cycle and finalizes the development of the software product.

3.3. Budget Administration

This section deals with the budget required to develop the product. Estimates are drawn, and they are checked, in the PPD review, to see if they are reasonable and if any updates are needed. For the purpose of this study, it is not discussed in detail.

## 4. Risk Analysis and Management

This section identifies the risks, the way to monitor them, and the corresponding actions to be taken if they occur. The list of factors that should be considered include contractual risks, technological risks, risks due to size and complexity of the product, risks in personnel acquisition and retention, and risks in achieving customer acceptance of the product. No risk factors were identified in our case.

## 5. Methods, Tools and Techniques

Methods for ensuring that all activities are carried out correctly are defined. These refer to reviews, audits and tests. In addition to the standards, and tools, if used.
In our case, the process used to develop the product is that of the Waterfall model. The requirement specification document is developed according to the standard developed for the Lebanese market. As to the other documents and plans-PDP, design specification document, and test plan-they are developed and conducted according to the corresponding IEEE or ISO standard. A tool is not needed to develop the software.

## 6. Statusing and Reporting

The engineering process is controlled by means of accurate status and reviews held at specific stages of the development. A regular weekly review of the project status as compared with the plan is required by project management. In addition, regular monthly reviews of project status vis-a-vis the plan are required.
If possible, quantifiable numbers reflecting the current amount of work completed are presented at these meetings and compared with the plan. These activities are scheduled for implementation.

## 7. Approvals

| | |
|---|---|
| Project Manager | Date |
| E.D.P. Manager | Date |
| Quality Manager | Date |

# Appendix C

# SDD for the Callback Billing System

## 1. Introduction

This section includes two sub-sections: scope and overview.

### 1.1. Scope

The design phase is the process which translates the requirement specification into a detailed representation of a software system.

### 1.2. Overview

This document translates the requirements of the billing system specified in the SSD into a detailed design of the system. The purpose is to design the software product architecture and its detailed design. The software is designed using structured analysis. The programming language used is MicroFocus COBOL.

## 2. System Architectural Design

The billing system is decomposed into many software programs. A brief description of the program's inputs, outputs and processing is presented only for the billing processing program.

2.1. CDR processing program

2.1.1. Inputs

Many input data is required for this program:

- CDR file containing the call records to be processed: CDR file
- parameter file containing the rate tables and area codes: PARAM file
- itemize file containing the processed calls on subscriber basis: ITEMIZE file
- subscribers file containing cumulative call information related to subscribers: SUB file
- history file containing log entry for every CDR file processed: HISTORY file

2.1.2. Processing

This program processes the records in the CDR file on a call-by-call basis.

Each record is analyzed to determine:

- Call type, whether calling card or call back which in turns determines the rate table to access in order to calculate call charges.
- Date and time the call was made.
- Identify the calling party and check for validity of subscriber.
- Calculate (leg a) in terms of minutes.
- Calculate (leg b) in terms of minutes.
- Analyze callback number and identify area code of originating country.
- Analyze destination number and identify area code of called country.

2.1.3. Outputs

No specific outputs are produced as a result from this program, however, many files are updated: SUB, ITEMIZE and HISTORY.

## 3. Detailed Design

The programs in the billing system are decomposed into several modules. A description of each module's inputs, outputs and processing is presented. Only the modules of the CDR processing program are described.

### 3.1.CDR processing program

#### 3.1.1. Duplication Module

One-Inputs

The first record of CDR is read, and a log entry is formatted as follows:

- Call date
- Call Time
- Calling Party
- Call Duration

Two- Processing

The log entry is checked for existence in the HISTORY file, if it does exist, this means the input CDR file is previously processed, the process is aborted, otherwise, the process is continued until end of input file.

Three- Outputs

In case CDR is previously processed a warning message is output to the billing operator, else a new log entry is output to the HISTORY file.

#### 3.1.2. Call Charging Module

One- Inputs

CDR records are input in sequence one after the other.

Two- Processing

call duration derivation of (leg a) and (leg b)

Initially, (leg b) is provided through duration field of the input record.

(leg a) is calculated as follows:

-both fields (calldisc time) and (in time) are determined in terms of seconds starting midnight

-if (in time) < (calldisc time), (leg a)=(calldisc time) - (in time)

otherwise the call started before midnight and ends on the next day in this case (leg a)=(calldisc time)+86400-(in time)

-the call is checked, if the first in a session then if it is a completed call then (leg a)=(leg a) - 90, otherwise, if it is an attempted call, then

- if (leg a) > 60 seconds then (leg a)=(leg a) - 60 otherwise (leg a) is set to zero

call charges

-callback number and destination numbers are analyzed in order to determine originating and terminating country codes, then these codes are used to access PARAM file to obtain (rate a) and (rate b) then:

call charge = (leg a) * (rate a) + (leg b) * (rate b)

Three- Outputs

A charged call record is written to the ITEMIZE file, and at the same time the SUB file is updated with the call charges for the related subscriber.

## 4. User Interface

User interface is kept to a minimum. For the CDR processing program, only one screen is designed, where the user operation is to provide and enter the CDR file name in a field in the screen and to hit the ENTER key for the billing program to run.

Error handling cases were taken into consideration as discussed in the detailed design. A field in the screen is reserved for the error message that is output in the case where the user enters a non-existent file or a duplicate file- a CDR file already processed.

## 5. Approvals

| | |
|---|---|
| Project Manager | Date |
| E.D.P. Manager | Date |
| Quality Manager | Date |

# Glossary

**AT** : Acceptance Test

**ATP** : Acceptance Test Procedure

**CC** : Configuration Control

**CDR** : Call Detail Record

**CDR** : Critical Design Review

**CM** : Configuration Management

**CMP** : Configuration Management Plan

**E.D.P.** : Electronic Data Processing

**FA** : Functional Audit

**GM** : General Manager

**ISO** : International Standards Organization

**OB** : Operational Baseline

**PA** : Physical Audit

**PB** : Product Baseline

**PDP** : Project Development Plan

**PDR** : Preliminary Design Review

**PM** : Project Manager

**POR** : Post Operation Review

**QA** : Quality Assurance

**QAS** : Quality Assurance System

**QM** : Quality Manager

**SCMP** : Software Configuration Management Plan

**SCMPR** : Software Configuration Management Plan Review

**SDD** : Software Design Description/document

**SQA** : Software Quality Assurance

**SQAP** : Software Quality Assurance Plans

**SRR** : Software Requirement Review

**SRS** : Software Requirement Specification

**SVVP** : Software Verification and Validation Plan

**SVVPR** : Software Verification and Validation Plan Review

**SVVR** : Software Verification and Validation Report

**TCM** : Test Coverage Metric

**TRM** : Test Result Metric

**UDR** : User Documentation Review

# References


**Bennatan. 1992**. Software Project Management: A Practitioner's Approach, McGraw-Hill.

**Fenton. 1993**. Software Metrics: A Rigorous Approach, Chapman and Hall.

**Gillies. 1992**. Software Quality: Theory and Management, Chapman and Hall.

**Hajjar. 1996**. Software Engineering Practices in Lebanon and Suggestions for Lebanese Requirements Analysis and Software Testing Standards.

**IEEE Standard 829-1983**. IEEE Standard for Software Test Documentation, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1008-1987**. IEEE Standard for Software Unit Test, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1016-1987**. IEEE Recommended Practice for Software Design Descriptions, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1016.1-1987**. IEEE Guide to Software Design Descriptions, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1058.1-1987**. IEEE Standard for Software Project Management Plan, Institute of Electrical and Electronics Engineers, Inc., New York.

**Ince. 1991**. Software Quality and Reliability, Chapman and Hall.

**Ince. 1994**. ISO 9001 and Software Quality Assurance, McGraw-Hill.

**ISO 9000-3. 1991**. Quality Management and Quality Assurance Standards-Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software. International Organization for Standardization (ISO).

**Johnson. 1993**. ISO 9000: Meeting the New International Standards, McGraw-Hill.

**Schmauch. 1994**. ISO 9000 for software developers, ASQC.

**Schulmeyer, McManus. 1988**. Handbook of Software Quality Assurance. Van Nostrand Reinhold Company.

**Smith. 1990**. Achieving Quality Software: Including its Application to Safety-Related Systems, Chapman and Hall.

**Vincent, Waters, Sinclair. 1988**. Software Quality Assurance, vol. 1, Prentice Hall.

**Vincent, Waters, Sinclair. 1988**. Software Quality Assurance, vol. 2, Prentice Hall.

**Wallmüller. 1994**. Software Quality Assurance: A practical Approach, Prentice Hall.

# Bibliography


**Australian Standards 1991-1993**. Published by Standards Australia, North Sydney.

**Bandinelli S., Fuggetta A. 1995**. Modeling and Improving an Industrial Software process, *IEEE Transactions on Software Engineering*, vol. 21, pp.440-468.

**Bennatan. 1992**. Software Project Management: A Practitioner's Approach, McGraw-Hill.

**Bersoff. 1984**. Elements of Software Configuration Management, Software Engineering , Vol.10, pp. 79-87.

**Bevan N. 1995**. Measuring Usability as Quality of Use, *Software Quality Journal*,4, pp.115-130.

**Dalal, Horgan, Kettenring. 1993**. Reliable Software and Communication: Software Quality, Reliability, and Safety, Institute of Electrical and Electronics Engineers.

**Debou C., Haux M., Jungmayr S. 1995**. A Measurement Framework for Improving Verification Processes, *Software Quality Journal*, 4. pp.207-225.

**Demarco. 1982**. Controlling Software Project, Yourdon Press.

**Fenton. 1993**. Software Metrics: A Rigorous Approach, Chapman and Hall.

**Fenton, Whitty, Iizuka. 1995**. Software Quality Assurance and Measurement: A Worldwide Perspective, International Thomson Computer Press.

**Freedman, Weinberg. 1984**. Reviews, Walkthroughs, and Inspections, Software Engineering , Vol.10, pp. 68-72.

**Frewin, Hatton. 1986**. Quality Management-Procedures and Practices, Software Engineering Journal, Vol.1, pp. 29-38.

**Gilb. 1988**. Principles of Software Engineering Management, Addison-Wesley.

**Gillies. 1992**. Software Quality: Theory and Management, Chapman and Hall.

**Hajjar. 1996**. Software Engineering Practices in Lebanon and Suggestions for Lebanese Requirements Analysis and Software Testing Standards.

**IEEE Software Engineering Standards Collection, Edition 1994**. The Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 828-1990**. IEEE Standard for Software Configuration Management Plans, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 829-1983**. IEEE Standard for Software Test Documentation, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1008-1987**. IEEE Standard for Software Unit Test, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1016-1987**. IEEE Recommended Practice for Software Design Descriptions, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1016.1-1987**. IEEE Guide to Software Design Descriptions, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1058.1-1987**. IEEE Standard for Software Project Management Plan, Institute of Electrical and Electronics Engineers, Inc., New York.

**IEEE Standard 1059-1993**. Guide for Verification and Validation Plans, Institute of Electrical and Electronics Engineers, Inc., New York.

**Ince. 1991**. Software Quality and Reliability, Chapman and Hall.

**Ince. 1994**. ISO 9001 and Software Quality Assurance, McGraw-Hill.

**ISO 9000-3. 1991**. Quality Management and Quality Assurance Standards-Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software. International Organization for Standardization (ISO).

**Johnson. 1993**. ISO 9000: Meeting the New International Standards, McGraw-Hill.

**Kitchenham, Fleeger. 1996**. Software Quality: The Elusive Target, *IEEE Software*, pp. 12-24.

**Lewin, Rosenau. 1988**. Software Project Management: Step by step, Marsha Lewin Associates, Inc. Publications.

**Ohmori A. 1993**. Software Quality Deployment Approach: Framework Design, Methodology and Example, *Software Quality Journal*, 3. pp. 209-240.

**Perry, Ermel, Shields. 1994**. Insider's Guide to Software Development, David Ewing.

**Saiedian. 1995**. SEI Capability Maturity Model's Impact on Contractors, Institute of Electrical and Electronics Engineers, Inc., New York.

**Schmauch. 1994**. ISO 9000 for software developers, ASQC.

**Schneidewind N. 1995**. Controlling and Predicting the Quality of Space Shuttle Software using Metrics, *Software Quality Journal*, 4. pp.49-68.

**Schulmeyer, McManus. 1988**. Handbook of Software Quality Assurance. Van Nostrand Reinhold Company.

**Smith. 1990**. Achieving Quality Software: Including its Application to Safety-Related Systems, Chapman and Hall.

**Vincent, Waters, Sinclair. 1988**. Software Quality Assurance, vol. 1, Prentice Hall.

**Vincent, Waters, Sinclair. 1988**. Software Quality Assurance, vol. 2, Prentice Hall.

**Wallace. 1989**. Software Verification and Validation: An Overview, Institute of Electrical and Electronics Engineers, Vol.6, pp. 10-17.

**Wallmüller. 1994**. Software Quality Assurance: A practical Approach, Prentice Hall.