chapter 1

The software quality challenge

Chapter outline

Two basic questions should be raised before we proceed to list the variety of subjects and details of the book:

(1) Is it justified to devote a special book to software quality assurance (SQA) or, in other words, can we not use the general quality assurance textbooks available that are applicable to numerous areas and industries?
(2) Having decided to develop specialized books for software quality assurance, at which of the various environments of software development, from amateurs' hobby to professionals' work, should we aim our main efforts? Put simply, what are the unique characteristics of the SQA environment?

The objective of this chapter is to answer these questions by exploring the related issues.

After completing this chapter, you will be able to:

- Identify the unique characteristics of software as a product and as production process that justify separate treatment of its quality issues.
- Recognize the characteristics of the environment where professional software development and maintenance take place.
- Explain the main environmental difficulties faced by software development and maintenance teams as a result of the environment in which they operate.

1.1 The uniqueness of software quality assurance

"Look at this," shouted my friend while handing me **Dagal Features**'s Limited Warranty leaflet. "Even **Dagal Features** can't cope with software bugs." He pointed to a short paragraph on page 3 of the leaflet that states the conditions of the warranty for **AMGAL**, a leading Software Master product sold all over the world. The leaflet states the following:

> LIMITED WARRANTY
> **Dagal Features** provides no warranty, either expressed or implied, with respect to **AMGAL**'s performance, reliability or fitness for any specified purpose. **Dagal Features** does not warrant that the software or its documentation will fulfil your requirements. although **Dagal Features** has performed thorough tests of the software and reviewed the documentation, **Dagal Features** does not provide any warranty that the software and its documentation are free of errors. **Dagal Features** will in no case be liable for any damages, incidental, direct, indirect or consequential, incurred as a result of impaired data, recovery costs, profit loss and third party claims. the software is licensed "as is". the purchaser assumes the complete risk stemming from application of the **AMGAL** program, its quality and performance.
>
> If physical defects are discovered in the documentation or the CD on which **AMGAL** is distributed, **Dagal Features** will replace, at no charge, the documentation or the CD within 180 days of purchase, provided proof of purchase is presented.

"Is the AMGAL software really so special that its developers are incapable of meeting the challenge of assuring a bug-free product?" continued my friend. "Do other software packages limit their warranties in the same way?"

Though **Dagal Features** and AMGAL are fictitious, an examination of the warranties offered by other software developers reveals a similar pattern. No developer will declare that its software is free of defects, as major manufacturers of computer hardware are wont to do. This refusal actually reflects the essential elemental differences between software and other industrial products, such as automobiles, washing machines or radios. These differences can be categorized as follows:

(1) **Product complexity**. Product complexity can be measured by the number of operational modes the product permits. An industrial product, even an advanced machine, does not allow for more than a few thousand modes of operation, created by the combinations of its different machine settings. Looking at a typical software package one can find millions of software operation possibilities. Assuring that the multitude of operational possibilities is correctly defined and developed is a major challenge to the software industry.

(2) **Product visibility**. Whereas the industrial products are visible, software products are invisible. Most of the defects in an industrial product can be detected during the manufacturing process. Moreover the absence of a part in an industrial product is, as a rule, highly visible (imagine a door missing from your new car). However, defects in software products (whether stored on diskettes or CDs) are invisible, as is the fact that parts of a software package may be absent from the beginning.

(3) **Product development and production process**. Let us now review the phases at which the possibility of detecting defects in an industrial product may arise:

   (a) **Product development**. In this phase the designers and quality assurance (QA) staff check and test the product prototype, in order to detect its defects.

   (b) **Product production planning**. During this phase the production process and tools are designed and prepared. In some products there is a need for a special production line to be designed and built. This phase thus provides additional opportunities to inspect the product, which may reveal defects that "escaped" the reviews and tests conducted during the development phase.

   (c) **Manufacturing**. At this phase QA procedures are applied to detect failures of products themselves. Defects in the product detected in the first period of manufacturing can usually be corrected by a change in the product's design or materials or in the production tools, in a way that eliminates such defects in products manufactured in the future.

In comparison to industrial products, software products do not benefit from the opportunities for detection of defects at all three phases of the production process. The only phase when defects can be detected is the development phase. Let us review what each phase contributes to the detection of defects:

(a) **Product development**. During this phase, efforts of the development teams and software quality assurance professionals are directed toward detecting inherent product defects. At the end of this phase an approved prototype, ready for reproduction, becomes available.

(b) **Product production planning**. This phase is not required for the software production process, as the manufacturing of software copies and printing of software manuals are conducted automatically. This applies to any software product, whether the number of copies is small, as in custom-made software, or large, as in software packages sold to the general public.

(c) **Manufacturing**. As mentioned previously, the manufacturing of software is limited to copying the product and printing copies of the software manuals. Consequently, expectations for detecting defects are quite limited during this phase.

The differences affecting the detection of defects in software products versus other industrial products are shown in Table 1.1 and Frame 1.1.

It should be noted that significant parts of advanced machinery as well as of household machines and other products include embedded software components (usually termed "firmware") that are integrated into the product. These software components (the firmware) share the same characteristics of the software products mentioned above. It follows that the comparison shown above should actually be that of software products versus other industrial products and non-software components of industrial products that include firmware. Hereinafter, when mentioning software, we will mean software products as well as firmware.

The fundamental differences between the development and production processes related to software products and those of other industrial products warrant the creation of a different SQA methodology for software. The need for special tools and methods for the software industry is reflected in the professional publications as well in special standards devoted to SQA, such as ISO 9000-3, "Guidelines for the application of ISO 9001 to the development, supply and maintenance of software". This point is supported by the fact that targeted guidelines have not been prepared by ISO for other industries,

Table 1.1: Factors affecting defect detection in software products vs. other industrial products

Characteristic	Software products	Other industrial products
Complexity	Usually, very complex product allowing for very large number of operational options	Degree of complexity much lower, allowing at most a few thousand operational options
Visibility of product	Invisible product, impossible to detect defects or omissions by sight (e.g. of a diskette or CD storing the software)	Visible product, allowing effective detection of defects by sight
Nature of development and production process	Opportunities to detect defects arise in only one phase, namely product development	Opportunities to detect defects arise in all phases of development and production: ■ Product development ■ Product production planning ■ Manufacturing

Frame 1.1 The uniqueness of the software development process

- **High complexity**, as compared to other industrial products
- **Invisibility of the product**
- **Opportunities to detect defects ("bugs")** are limited to the product development phase

and the only other targeted guidelines have been prepared for services (ISO 9004-2, "Quality management and quality systems elements: Guidelines for the services").

The great complexity as well as invisibility of software, among other product characteristics, make the development of SQA methodology and its successful implementation a highly professional challenge.

1.2 The environments for which SQA methods are developed

The software developed by many individuals and in different situations fulfills a variety of needs:

- Pupils and students develop software as part of their education.
- Software amateurs develop software as a hobby.
- Professionals in engineering, economics, management and other fields develop software to assist them in their work, to perform calculations, summarize research and survey activities, and so forth.
- Software development professionals (systems analysts and programmers) develop software products or firmware as a professional career objective while in the employment of software houses or by software development and maintenance units (teams, departments, etc.) of large and small industrial, financial and other organizations.

All those who participate in these activities are required to deal with software quality problems ("bugs"). However, quality problems in their most severe form govern the professional software development.

This book is devoted, therefore, to defining and solving many of the software quality assurance (SQA) problems confronted by software development and maintenance professionals. However, all other types of software developers can find portions of the book applicable to and recommended for their own software development efforts.

Let us begin with the examination of the environment of professional software development and maintenance (hereafter "the SQA environment"), as it is a major consideration in the development of SQA methodologies and their implementation. The main characteristics of this environment are as follows:

(1) **Contractual conditions**. As a result of the commitments and conditions defined in the contract between the software developer and the customer, the activities of software development and maintenance need to cope with:
   - A defined list of functional requirements that the developed software and its maintenance need to fulfill.
   - The project budget.
   - The project timetable.

The managers of software development and maintenance projects need to invest a considerable amount of effort in the oversight of activities in order to meet the contract's requirements.

(2) **Subjection to customer–supplier relationship**. Throughout the process of software development and maintenance, activities are under the oversight of the customer. The project team has to cooperate continuously with the customer: to consider his request for changes, to discuss his criticisms about the various aspects of the project, and to get his approval for changes initiated by the development team. Such relationships do not usually exist when software is developed by non-software professionals.

(3) **Required teamwork.** Three factors usually motivate the establishment of a project team rather than assigning the project to one professional:

- Timetable requirements. In other words, the workload undertaken during the project period requires the participation of more than one person if the project is to be completed on time.
- The need for a variety of specializations in order to carry out the project.
- The wish to benefit from professional mutual support and review for the enhancement of project quality.

(4) **Cooperation and coordination with other software teams**. The carrying-out of projects, especially large-scale projects, by more than one team is a very common event in the software industry. In these cases, cooperation may be required with:

- Other software development teams in the same organization.
- Hardware development teams in the same organization.
- Software and hardware development teams of other suppliers.
- Customer software and hardware development teams that take part in the project's development.

An outline of cooperation needs, as seen from the perspective of the development team, is shown in Figure 1.1.

(5) **Interfaces with other software systems**. Nowadays, most software systems include interfaces with other software packages. These interfaces allow data in electronic form to flow between the software systems, free from keying in of data processed by the other software systems. One can identify the following main types of interfaces:

- Input interfaces, where other software systems transmit data to your software system.
- Output interfaces, where your software system transmits processed data to other software systems.
- Input and output interfaces to the machine's control board, as in medical and laboratory control systems, metal processing equipment, etc.

Salary processing software packages provide good examples of typical input and output interfaces to other software packages – see Figure 1.2. First let us look at the input interface. In order to calculate salaries, one needs the employees' attendance information, as captured by the time

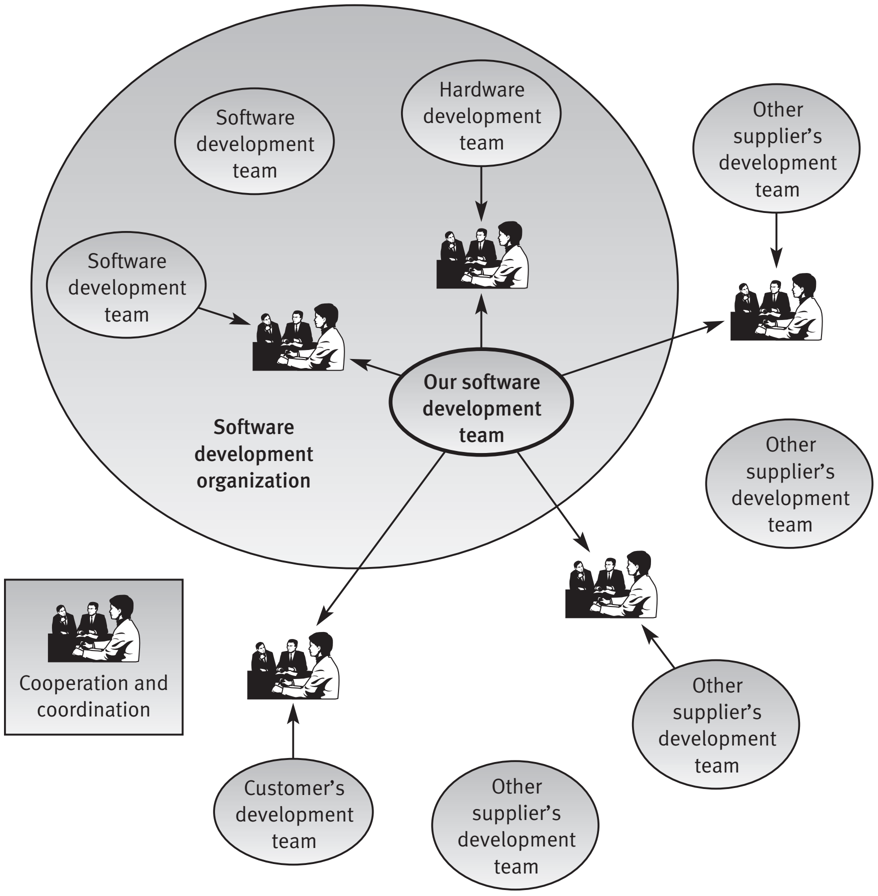
Hardware development team
Software development team
Other supplier's development team
Software development team
Our software development team
Software development organization
Other supplier's development team
Cooperation and coordination
Other supplier's development team
Customer's development team
Other supplier's development team

Figure 1.1: A cooperation and coordination scheme for a software development team of a large-scale project

clocks placed at the entrance to the office building and processed later by the attendance control software system. Once a month, this information (the attendance lists including the overtime data) is transmitted electronically from the attendance control system to the salary processing system. This information transmission represents an input interface for the salary processing software system; at the same time it represents an output interface to the attendance control system. Now, let us examine the output interface of our system. One of the outputs of the salary processing system is the list of "net" salaries, after deduction of the income tax and other items, payable to the employees. This list, including the employees' bank account details, has to be sent to the banks. The transmission of the list of salary payments is done electronically, representing an output interface for the salary processing system and an input interface for the bank's account system.

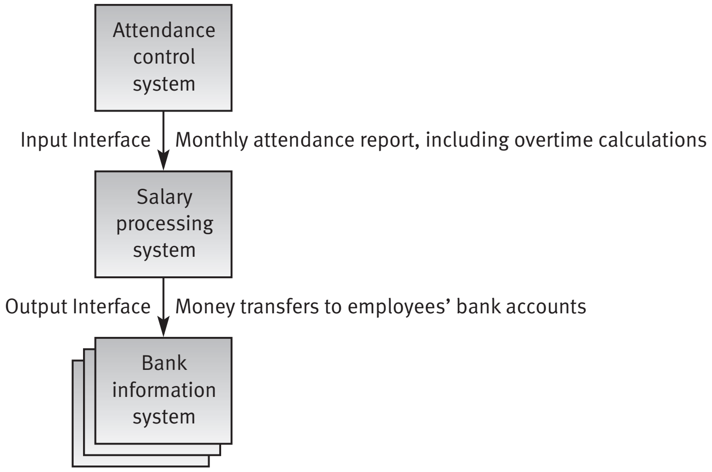
Attendance control system
Input Interface
Monthly attendance report, including overtime calculations
Salary processing system
Output Interface
Money transfers to employees' bank accounts
Bank information system

Figure 1.2: The salary software system – an example of software interfaces

(6) **The need to continue carrying out a project despite team member changes.** It is quite common for team members to leave the team during the project development period, whether owing to promotions to higher level jobs, a switch in employers, transfers to another city, and so forth. The team leader then has to replace the departing team member either by another employee or by a newly recruited employee. No matter how much effort is invested in training the new team member, "the show must go on", which means that the original project contract timetable will not change.

(7) **The need to continue carrying out software maintenance for an extended period.** Customers who develop or purchase a software system expect to continue utilizing it for a long period, usually for 5–10 years. During the service period, the need for maintenance will eventually arise. In most cases, the developer is required to supply these services directly. Internal "customers", in cases where the software has been developed in-house, share the same expectation regarding the software maintenance during the service period of the software system.

The environmental characteristics create a need for intensive and continuous managerial efforts parallel to the professional efforts that have to be invested in order to assure the project's quality, in other words to assure the project's success.

A summary of the main characteristics of the SQA environment is shown in Frame 1.2.

A significant amount of software as well as firmware development is not carried out subject to formal contracts or formal customer–supplier relationships, as mentioned in the first two SQA environment characteristics. This type of activity usually concerns software developed in-house for internal use

chapter 3

Software quality factors

Chapter outline

We have already established (see Chapter 2) that the requirements document is one of the most important elements for achieving software quality. Here we ask: What is a "good" software requirements document? We want to explore what subjects and aspects of software use should be covered in the document.

This chapter is, therefore, dedicated to the review of the wide spectrum of aspects of software use that may be operative throughout the life cycle of software systems. Some SQA models suggest that the wide spectrum of requirements should be classified into 11 to 15 factors (subject areas) that can be amalgamated into three or four categories.

After completing this chapter, you will be able to:

- Explain the need for comprehensive requirements documents and characterize the contents of such documents.
- Explain the structure (categories and factors) of McCall's classic factor model.

- List the factors, other than those included in McCall's model, that are suggested by the alternative SQA models.
- Identify who is interested in the definition of quality requirements.

3.1 The need for comprehensive software quality requirements

- "Our new sales information system seems okay, the invoices are correct, the inventory records are correct, the discounts granted to our clients exactly follow our very complicated discount policy, **but** our new sales information system frequently fails, usually at least twice a day, each time for twenty minutes or more. Yesterday it took an hour and half before we could get back to work . . . . Imagine how embarrassing it is to store managers . . . . Softbest, the software house that developed our computerized sales system, claims no responsibility . . . ."
- "Just half a year ago we launched our new product – the radar detector. The firmware RD-8.1, embedded in this product, seems to be the cause for its success. **But**, when we began planning the development of a European version of the product, we found out that though the products will be almost similar, our software development department needs to develop new firmware; almost all the design and programming will be new."
- "Believe it or not, our software package 'Blackboard' for schoolteachers, launched just three months ago, is already installed in 187 schools. The development team just returned from a week in Hawaii, their vacation bonus. **But** we have been suddenly receiving daily complaints from the 'Blackboard' maintenance team. They claim that the lack of failure-detection features in the software, in addition to the poor programmer's manual, have caused them to invest more than the time estimated to deal with bugs or adding minor software changes that were agreed as part of purchasing contracts with clients."
- "The new version of our loan contract software is really accurate. We have already processed 1200 customer requests, and checked each of the output contracts. There were no errors. **But** we did face a severe unexpected problem – training a new staff member to use this software takes about two weeks. This is a real problem in customers' departments suffering from high employee turnover . . . . The project team says that as they were not required to deal with training issues in time, an additional two to three months of work will be required to solve the problem."

There are some characteristic common to all these "but's":

- All the software projects satisfactorily fulfilled the basic requirements for correct calculations (correct inventory figures, correct average class's score, correct loan interest, etc.).
- All the software projects suffered from poor performance in important areas such as maintenance, reliability, software reuse, or training.

- The cause for the poor performance of the developed software projects in these areas was the lack of predefined requirements to cover these important aspects of the software's functionality.

The need for a comprehensive definition of requirements

There is a need for a comprehensive definition of requirements that will cover all attributes of software and aspects of the use of software, including usability aspects, reusability aspects, maintainability aspects, and so forth in order to assure the full satisfaction of the users.

The great variety of issues related to the various attributes of software and its use and maintenance, as defined in software requirements documents, can be classified into content groups called *quality factors*. We expect the team responsible for defining the software requirements of a software system to examine the need to define the requirements that belong to each factor. Software requirement documents are expected to differ in the emphasis placed on the various factors, a reflection of the differences to be found among software projects. Thus, we can expect that not all the factors will be universally "represented" in all the requirements documents.

The next sections deal with the classification of quality requirements into quality factors. Obviously, only the major approaches to this topic will be covered.

3.2 Classifications of software requirements into software quality factors

Several models of software quality factors and their categorization in factor categories have been suggested over the years. The classic model of software quality factors, suggested by McCall, consists of 11 factors (McCall *et al.*, 1977). Subsequent models, consisting of 12 to 15 factors, were suggested by Deutsch and Willis (1988) and by Evans and Marciniak (1987). The alternative models do not differ substantially from McCall's model. The McCall factor model, despite the quarter of a century of its "maturation", continues to provide a practical, up-to-date method for classifying software requirements (Pressman, 2000).

McCall's factor model

McCall's factor model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision and product transition – as follows:

- **Product operation factors:** Correctness, Reliability, Efficiency, Integrity, Usability.
- **Product revision factors:** Maintainability, Flexibility, Testability.
- **Product transition factors:** Portability, Reusability, Interoperability.

McCall's model and its categories are illustrated by the McCall model of software quality factors tree (see Figure 3.1).

The next three sections are dedicated to a detailed description of the software quality factors included in each of McCall's categories.

3.3 Product operation software quality factors

According to McCall's model, five software quality factors are included in the product operation category, all of which deal with requirements that directly affect the daily operation of the software. These factors are as follows.

Correctness

Correctness requirements are defined in a list of the software system's required outputs, such as a query display of a customer's balance in the sales accounting information system, or the air supply as a function of temperature specified by the firmware of an industrial control unit. Output specifications are usually multidimensional; some common dimensions include:

- The output mission (e.g., sales invoice printout, and red alarms when temperature rises above 250°F).

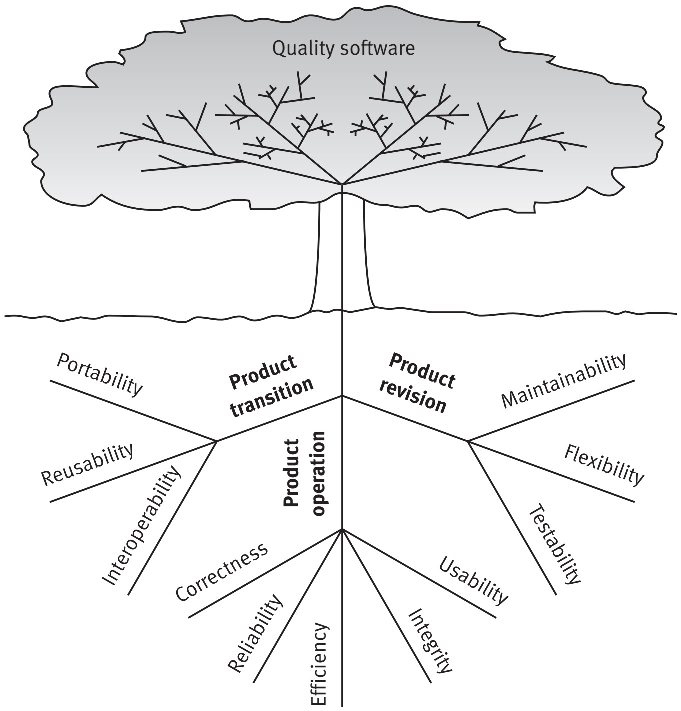

Figure 3.1: McCall's factor model tree

Source: Based on McCall *et al.*, 1977

- The required accuracy of those outputs that can be adversely affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be adversely affected by incomplete data.
- The up-to-dateness of the information (defined as the time between the event and its consideration by the software system).
- The availability of the information (the reaction time, defined as the time needed to obtain the requested information or as the requested reaction time of the firmware installed in a computerized apparatus).
- The standards for coding and documenting the software system.

Example

The correctness requirements of a club membership information system consisted of the following:

- The output mission: A defined list of 11 types of reports, four types of standard letters to members and eight types of queries, which were to be displayed on the monitor on request.
- The required accuracy of the outputs: The probability for a non-accurate output, containing one or more mistakes, will not exceed 1%.
- The completeness of the output information: The probability of missing data about a member, his attendance at club events, and his payments will not exceed 1%.
- The up-to-dateness of the information: Not more than two working days for information about participation in events and not more than one working day for information about entry of member payments and personal data.
- The availability of information: Reaction time for queries will be less than two seconds on average; the reaction time for reports will be less than four hours.
- The required standards and guidelines: The software and its documentation are required to comply with the client's guidelines.

Reliability

Reliability requirements deal with failures to provide service. They determine the maximum allowed software system failure rate, and can refer to the entire system or to one or more of its separate functions.

Examples

(1) The failure frequency of a heart-monitoring unit that will operate in a hospital's intensive care ward is required to be less than one in 20 years. Its heart attack detection function is required to have a failure rate of less than one per million cases.

(2) One requirement of the new software system to be installed in the main branch of Independence Bank, which operates 120 branches, is that it will not fail, on average, more than 10 minutes per month during the bank's office hours. In addition, the probability that the off-time (the time needed for repair and recovery of all the bank's services) be more than 30 minutes is required to be less than 0.5%.

Efficiency

Efficiency requirements deal with the hardware resources needed to perform all the functions of the software system in conformance to all other requirements. The main hardware resources to be considered are the computer's processing capabilities (measured in MIPS – million instructions per second, MHz or megahertz – million cycles per second, etc.), its data storage capability in terms of memory and disk capacity (measured in MBs – megabytes, GBs – gigabytes, TBs – terabytes, etc.) and the data communication capability of the communication lines (usually measured in KBPS – kilobits per second, MBPS – megabits per second, and GBPS – gigabits per second). The requirements may include the maximum values at which the hardware resources will be applied in the developed software system or the firmware.

Another type of efficiency requirement deals with the time between recharging of the system's portable units, such as, information systems units located in portable computers, or meteorological units placed outdoors.

Examples

(1) A chain of stores is considering two alternative bids for a software system. Both bids consist of placing the same computers in the chain's headquarters and its branches. The bids differ solely in the storage volume: 20 GB per branch computer and 100 GB in the head office computer (Bid A); 10 GB per branch computer and 30 GB in the head office computer (Bid B). There is also a difference in the number of communication lines required: Bid A consists of three communication lines of 28.8 KBPS between each branch and the head office, whereas Bid B is based on two communication lines of the same capacity between each branch and the head office. In this case, it is clear that Bid B is more efficient than Bid A because fewer hardware resources are required.

(2) An outdoor meteorological unit, equipped with a 1000 milli-ampere hour cell, should be capable of supplying the power requirements of the unit for at least 30 days. The system performs measurements once per hour, logs the results, and transmits the results once a day to the meteorological center by means of wireless communication.

Integrity

Integrity requirements deal with the software system security, that is, requirements to prevent access to unauthorized persons, to distinguish between the majority of personnel allowed to see the information ("read permit") and a limited group who will be allowed to add and change data ("write permit"), and so forth.

Example
The Engineering Department of a local municipality operates a GIS (Geographic Information System). The Department is planning to allow citizens access to its GIS files through the Internet. The software requirements include the possibility of viewing and copying but not inserting changes in the maps of their assets as well as any other asset in the municipality's area ("read only" permit). Access will be denied to plans in progress and to those maps defined by the Department's head as limited access documents.

Usability
Usability requirements deal with the scope of staff resources needed to train a new employee and to operate the software system. For more about usability see Juristo *et al.* (2001), Donahue (2001) and Ferre *et al.* (2001).

Example
The software usability requirements document for the new help desk system initiated by a home appliance service company lists the following specifications:

(a) A staff member should be able to handle at least 60 service calls a day.
(b) Training a new employee will take no more than two days (16 training hours), immediately at the end of which the trainee will be able to handle 45 service calls a day.

3.4 Product revision software quality factors

According to the McCall model of software quality factors, three quality factors comprise the product revision category. These factors deal with those requirements that affect the complete range of software maintenance activities: corrective maintenance (correction of software faults and failures), adaptive maintenance (adapting the current software to additional circumstances and customers without changing the software) and perfective maintenance (enhancement and improvement of existing software with respect to locally limited issues). These are as follows.

Maintainability
Maintainability requirements determine the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections. This factor's requirements refer to the modular structure of software, the internal program documentation, and the programmer's manual, among other items.

Example
Typical maintainability requirements:

(a) The size of a software module will not exceed 30 statements.
(b) The programming will adhere to the company coding standards and guidelines.

Flexibility
The capabilities and efforts required to support adaptive maintenance activities are covered by the flexibility requirements. These include the resources (i.e. in man-days) required to adapt a software package to a variety of customers of the same trade, of various extents of activities, of different ranges of products and so on. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.

Example
TSS (teacher support software) deals with the documentation of pupil achievements, the calculation of final grades, the printing of term grade documents, and the automatic printing of warning letters to parents of failing pupils. The software specifications included the following flexibility requirements:

(a) The software should be suitable for teachers of all subjects and all school levels (elementary, junior and high schools).
(b) Non-professionals should be able to create new types of reports according to the schoolteacher's requirements and/or the city's education department demands.

Testability
Testability requirements deal with the testing of an information system as well as with its operation. Testability requirements for the ease of testing are related to special features in the programs that help the tester, for instance by providing predefined intermediate results and log files. Testability requirements related to software operation include automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the software system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

Example
An industrial computerized control unit is programmed to calculate various measures of production status, report the performance level of the machinery, and operate a warning signal in predefined situations. One testability

requirement demanded was to develop a set of standard test data with known system expected correct reactions in each stage. This standard test data is to be run every morning, before production begins, to check whether the computerized unit reacts properly.

3.5 Product transition software quality factors

According to McCall, three quality factors are included in the product transition category, a category that pertains to the adaptation of software to other environments and its interaction with other software systems.

Portability
Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. These requirements make it possible to continue using the same basic software in diverse situations or to use it simultaneously in diverse hardware and operating systems situations.

Example
A software package designed and programmed to operate in a Windows 2000 environment is required to allow low-cost transfer to Linux and Windows NT environments.

Reusability
Reusability requirements deal with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules. These benefits of higher quality are based on the assumption that most of the software faults have already been detected by the quality assurance activities performed on the original software, by users of the original software, and during its earlier reuses. The issues of software reuse became a subject of software industry standards (see IEEE, 1999).

Example
A software development unit has been required to develop a software system for the operation and control of a hotel swimming pool that serves hotel guests and members of a pool club. Although the management did not define any reusability requirements, the unit's team leader, after analyzing the information processing requirements of the hotel's spa, decided to add the reusability requirement that some of the software modules for the pool should be designed and programmed in a way that will allow its reuse in the spa's future software system, which is planned to be developed next year.

These modules will allow:

- Entrance validity checks of membership cards and visit recording.
- Restaurant billing.
- Processing of membership renewal letters.

Interoperability
Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware (for example, the firmware of the production machinery and testing equipment interfaces with the production control software). Interoperability requirements can specify the name(s) of the software or firmware for which interface is required. They can also specify the output structure accepted as standard in a specific industry or applications area.

Example
The firmware of a medical laboratory's equipment is required to process its results (output) according to a standard data structure that can then serve as input for a number of standard laboratory information systems.

3.6 Alternative models of software quality factors

Two factor models, appearing during the late 1980s, considered to be alternatives to the McCall classic factor model (McCall *et al*., 1977), deserve discussion:

- The Evans and Marciniak factor model (Evans and Marciniak, 1987).
- The Deutsch and Willis factor model (Deutsch and Willis, 1988).

3.6.1 Formal comparison of the alternative models

A formal comparison of the factor models reveals:

- Both alternative models exclude only one of McCall's 11 factors, namely the testability factor.
- The Evans and Marciniak factor model consists of 12 factors that are classified into three categories.
- The Deutsch and Willis factor model consists of 15 factors that are classified into four categories.

Taken together, five new factors were suggested by the two alternative factor models:

- Verifiability (by both models)
- Expandability (by both models)

- Safety (by Deutsch and Willis)
- Manageability (by Deutsch and Willis)
- Survivability (by Deutsch and Willis).

The factors included in the various factor models are compared in Table 3.1. The additional factors are defined as follows.

Verifiability (suggested by Evans and Marciniak)

Verifiability requirements define design and programming features that enable efficient verification of the design and programming. Most verifiability requirements refer to modularity, to simplicity, and to adherence to documentation and programming guidelines.

Expandability (suggested by Evans and Marciniak, and Deutsch and Willis)

Expandability requirements refer to future efforts that will be needed to serve larger populations, improve service, or add new applications in order to improve usability. The majority of these requirements are covered by McCall's flexibility factor.

Safety (suggested by Deutsch and Willis)

Safety requirements are meant to eliminate conditions hazardous to operators of equipment as a result of errors in process control software. These errors can result in inappropriate reactions to dangerous situations or to the failure to provide alarm signals when the dangerous conditions to be detected by the software arise.

Table 3.1: Comparison of McCall's factor model and alternative models

			Alternative factor models	
No.	Software quality factor	McCall's classic model	Evans and Marciniak	Deutsch and Willis
1	Correctness	+	+	+
2	Reliability	+	+	+
3	Efficiency	+	+	+
4	Integrity	+	+	+
5	Usability	+	+	+
6	Maintainability	+	+	+
7	Flexibility	+	+	+
8	Testability	+		
9	Portability	+	+	+
10	Reusability	+	+	+
11	Interoperability	+	+	+
12	Verifiability		+	+
13	Expandability		+	+
14	Safety			+
15	Manageability			+
16	Survivability			+

Example
In a chemical plant, a computerized system controls the flow of acid according to pressure and temperature changes occurring during production. The safety requirements refer to the system's computerized reactions in cases of dangerous situations and also specify what kinds of alarms are needed in each case.

Manageability (suggested by Deutsch and Willis)
Manageability requirements refer to the administrative tools that support software modification during the software development and maintenance periods, such as configuration management, software change procedures, and the like.

Example
"Chemilog" is a software system that automatically logs the flows of chemicals into various containers to allow for later analysis of the efficiency of production units. The development and issue of new versions and releases of "Chemilog" are controlled by the Software Development Board, whose members act according to the company's software modifications procedure.

Survivability (suggested by Deutsch and Willis)
Survivability requirements refer to the continuity of service. These define the minimum time allowed between failures of the system, and the maximum time permitted for recovery of service, two factors that pertain to service continuity. Although these requirements may refer separately to total and to partial failures of services, they are especially geared to failures of essential functions or services. Significant similarity exists between the survivability factor and the reliability factor described in McCall's model.

Example
Taya operates a national lottery, held once a week. About 400 000 to 700 000 bets are placed weekly. The new software system the customer (the Taya National Lottery) has ordered will be highly computerized and based on a communication system that connects all the betting machines to the central computer. To its other high reliability requirements, Taya has added the following survivability requirement: The probability that unrecoverable damage to the betting files will occur in case of any system failure is to be limited to less than one in a million.

3.6.2 Comparison of the factor models – content analysis

After comparing the contents of the factor models, we find that two of the five additional factors, Expandability and Survivability, actually resemble factors already included in McCall's factor model, though under different names, Flexibility and Reliability. In addition, McCall's Testability factor can be considered as one element in his own Maintainability factor.

This implies that the differences between the three factor models are much smaller than initially perceived. That is, the alternative factor models add only three "new" factors to McCall's model:

- Both models add the factor Verifiability.
- The Deutsch and Willis model adds the factors Safety and Manageability.

3.6.3 Structure of the alternative factor models

Nevertheless, despite their similarities, the categories employed by the alternative factor models and the classification of the specific factors into these categories differ from those offered by McCall's model. Table 3.2 compares the structure of the three models according to the factors and their classification into the categories.

3.7 Who is interested in the definition of quality requirements?

Naturally, one might think that only the client is interested in thoroughly defining his requirements in order to assure the quality of the software product he contracted. The requirements document he prepares does indeed serve as a fundamental protection against low quality. However, our analysis of the various quality factors indicates how the software developer can add requirements that represent his own interest. Following are some examples:

(1) **Reusability requirements**. In cases where the client anticipates development in the near future of an additional software system having strong similarities to the present software, the client will himself initiate reusability requirements. In other cases, the client is interested in reusing parts of software systems that were developed earlier in a new system. However, it is more likely that the developer, who serves a great variety of clients, will recognize the potential benefits of reuse, and will enter reusability into the list of requirements to be fulfilled by the project team.

(2) **Verifiability requirements**. These requirements are meant to improve the design reviews and software tests carried out during software development. Their aim is to save development resources and they are, therefore, of interest to developers. The client, however, is usually uninterested in placing requirements that deal with the internal activities of the developer team.

Some quality factors not included in the typical client's requirements document may, in many cases, interest the developer. The following list of quality factors usually interest the developer whereas they may raise very little interest on the part of the client:

- Portability
- Reusability
- Verifiability.

Table 3.2: Comparison of the structure of McCall's factor model *vis-à-vis* the three alternative models

McCall's model categories	Software quality factors	Deutsch and Willis model categories				Evans and Marciniak model categories		
		Functional	Performance	Change	Management	Design	Performance	Adaptation
Product operation	Correctness		x			x		
	Reliability	x					x	
	Efficiency		x				x	
	Integrity	x					x	
	Usability	x					x	
Product revision	Maintainability			x		x		
	Flexibility			x				x
	Testability							
Product transition	Portability			x				x
	Reusability			x				x
	Interoperability		x					x
Factors of the alternative models	Verifiability				x	x		
	Expandability			x				x
	Safety		x					
	Manageability				x			
	Survivability	x						

So, one can expect that a project will be carried out according to two requirements documents:

- The client's requirements document
- The developer's additional requirements document.

3.8 Software compliance with quality factors

Throughout the software development process, the extent to which the process complies with the requirements of the various quality factors is examined by design reviews, software inspections, software tests, and so forth. Comprehensive discussions of design reviews, software testing, software quality metrics and other tools for verifying and validating the quality of software are provided in the balance of this book.

Furthermore, the software product's compliance to the requirements belonging to the various quality factors is measured by software quality metrics, measures that quantify the degree of compliance. In order to allow for valid measurements of compliance, sub-factors have been defined for those quality factors that represent a wide range of attributes and aspects of software use. Software quality metrics are suggested for each of these sub-factors. Chapter 21 is dedicated to the subject of software metrics.

Table 3.3 presents some of these sub-factors, the majority of which were suggested by Evans and Marciniak (1987).

Table 3.3: Factors and sub-factors

Factor model	Software quality factors	Sub-factors
McCall's model: Product operation category	Correctness	Accuracy Completeness Up-to-dateness Availability (response time) Coding and documentation guidelines compliance (consistency)
	Reliability	System reliability Application reliability Computational failure recovery Hardware failure recovery
	Efficiency	Efficiency of processing Efficiency of storage Efficiency of communication Efficiency of power usage (for portable units)
	Integrity	Access control Access audit
	Usability	Operability Training

▶

chapter 4

The components of the software quality assurance system – overview

Chapter outline

This chapter, the final chapter of the introductory portion of the text, is dedicated to a schematic overview of the wide range of SQA components available to planners of an intra-organizational SQA system. As a local system, an intra-organizational SQA system bears "local colors", which are affected by the characteristics of the organization, its development projects, software maintenance activities, and professional staff. The concise description of SQA components is followed by a discussion of the considerations guiding construction of an organization's SQA system. This glimpse will allow you to obtain some preliminary understanding about the potential contribution of each component, about the entire range of components, and about the system as a defined entity.

4.1 The SQA system – an SQA architecture

An SQA system always combines a wide range of SQA components, all of which are employed to challenge the multitude of sources of software errors and to achieve an acceptable level of software quality. As stated in Chapter 1, the task of SQA is unique in the area of quality assurance due to the special characteristics of software. In addition, the environment in which software development and maintenance is undertaken directly influences the SQA components (see Chapter 1).

SQA system components can be classified into six classes:

- **Pre-project components**. To assure that (a) the project commitments have been adequately defined considering the resources required, the schedule and budget; and (b) the development and quality plans have been correctly determined.
- **Components of project life cycle activities assessment**. The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage.

  The development life cycle stage components detect design and programming errors. Its components are divided into the following four sub-classes:

  – Reviews
  – Expert opinions
  – Software testing.

  The SQA components used during the operation–maintenance phase include specialized maintenance components as well as development life cycle components, which are applied mainly for functionality improving maintenance tasks.

  An additional sub-class of SQA project life cycle components deals with assuring the quality of project parts performed by subcontractors and other external participants during project development and maintenance.
- **Components of infrastructure error prevention and improvement**. The main objectives of these components, which are applied throughout the

entire organization, are to eliminate or at least reduce the rate of errors, based on the organization's accumulated SQA experience.

- **Components of software quality management.** This class of components is geared toward several goals, the major ones being the control of development and maintenance activities and the introduction of early managerial support actions that mainly prevent or minimize schedule and budget failures and their outcomes.
- **Components of standardization, certification, and SQA system assessment.** These components implement international professional and managerial standards within the organization. The main objectives of this class are (a) utilization of international professional knowledge, (b) improvement of coordination of the organizational quality systems with other organizations, and (c) assessment of the achievements of quality systems according to a common scale. The various standards may be classified into two main groups: (a) quality management standards, and (b) project process standards.
- **Organizing for SQA – the human components.** The SQA organizational base includes managers, testing personnel, the SQA unit and practitioners interested in software quality (SQA trustees, SQA committee members and SQA forum members). All these *actors* contribute to software quality; their main objectives are to initiate and support the implementation of SQA components, detect deviations from SQA procedures and methodology, and suggest improvements.

The entire range of SQA system components by its classes is presented in Frame 4.1.

Frame 4.1 SQA system component classes

Pre-project quality components

Project life cycle quality components

Infrastructure error preventive and improvement components

Software quality management components

Standardization, certification and SQA assessment components

Organizing for SQA – the human components

The spectrum of SQA components presented in this book reflects the comprehensive conception of SQA adopted by the author (see Frame 2.6). Accordingly, several of the SQA components presented here are unique to this volume, and not found in other SQA texts.

A graphic illustration of SQA system components as the SQA architecture is presented in Figure 4.1. Included are references to the chapters that discuss each component in detail. An overview of the system immediately follows.

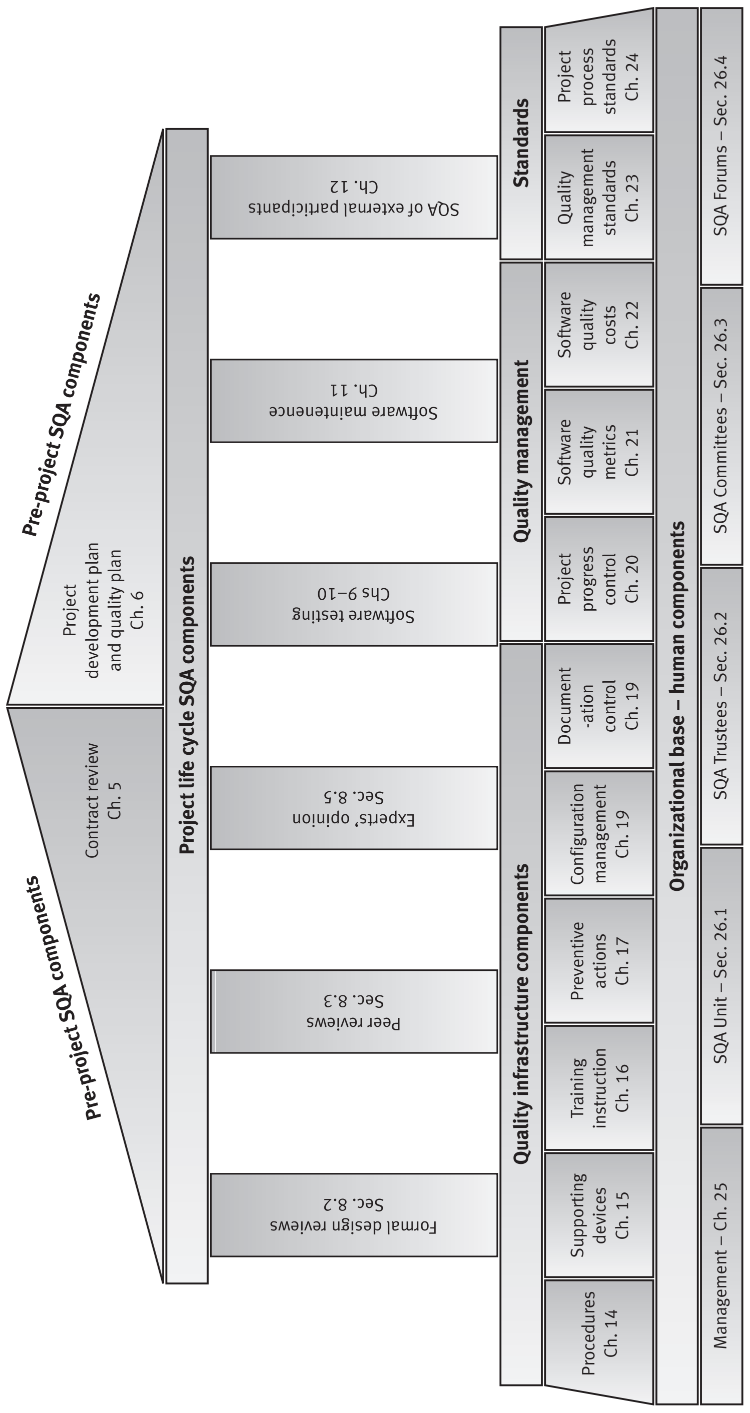
Pre-project SQA components
Pre-project SQA components
Contract review Ch. 5
Project development plan and quality plan Ch. 6
Project life cycle SQA components
Formal design reviews Sec. 8.2
Peer reviews Sec. 8.3
Experts' opinion Sec. 8.5
Software testing Chs 9–10
Software maintenence Ch. 11
SQA of external participants Ch. 12
Quality infrastructure components
Quality management
Standards
Procedures Ch. 14
Supporting devices Ch. 15
Training instruction Ch. 16
Preventive actions Ch. 17
Configuration management Ch. 19
Document -ation control Ch. 19
Project progress control Ch. 20
Software quality metrics Ch. 21
Software quality costs Ch. 22
Quality management standards Ch. 23
Project process standards Ch. 24
Organizational base – human components
Management – Ch. 25
SQA Unit – Sec. 26.1
SQA Trustees – Sec. 26.2
SQA Committees – Sec. 26.3
SQA Forums – Sec. 26.4

Figure 4.1: "The software quality shrine" – the SQA architecture

4.2 Pre-project components

The SQA components belonging here are meant to improve the preparatory steps taken prior to initiating work on the project itself:

- Contract review
- Development and quality plans.

4.2.1 Contract review

Software may be developed within the framework of a contract negotiated with a customer or in response to an internal order originating in another department. An internal order may entail a request for developing a firmware software system to be embedded within a hardware product, an order for a software product to be sold as a package, or an order for the development of administrative software to be applied within the company. In all these instances, the development unit is committed to an agreed-upon functional specification, budget and schedule.

Accordingly, contract review activities must include a detailed examination of (a) the project proposal draft and (b) the contract drafts. Specifically, contract review activities include:

- Clarification of the customer's requirements
- Review of the project's schedule and resource requirement estimates
- Evaluation of the professional staff's capacity to carry out the proposed project
- Evaluation of the customer's capacity to fulfill his obligations
- Evaluation of development risks.

A similar approach is applied in the review of maintenance contracts. Such reviews take into account that besides error corrections, maintenance services include software adaptation and limited software development activities for the sake of performance improvement (termed "functionality improvement maintenance").

4.2.2 Development and quality plans

Once a software development contract has been signed or a commitment made to undertake an internal project for the benefit of another department of the organization, a plan is prepared of the project ("development plan") and its integrated quality assurance activities ("quality plan"). These plans include additional details and needed revisions based on prior plans that provided the basis for the current proposal and contract. It is quite common for several months to pass between the tender submission and the signing of the contract. During this period, changes may occur in staff availability, in professional capabilities, and so forth. The plans are then revised to reflect the changes that occurred in the interim.

The main issues treated in the project development plan are:

- Schedules
- Required manpower and hardware resources
- Risk evaluations
- Organizational issues: team members, subcontractors and partnerships
- Project methodology, development tools, etc.
- Software reuse plans.

The main issues treated in the project's quality plan are:

- Quality goals, expressed in the appropriate measurable terms
- Criteria for starting and ending each project stage
- Lists of reviews, tests, and other scheduled verification and validation activities.

4.3 Software project life cycle components

The project life cycle is composed of two stages: the development life cycle stage and the operation–maintenance stage.

Several SQA components enter the software development project life cycle at different points. Their use should be planned prior to the project's initiation. The main components are:

- Reviews
- Expert opinions
- Software testing
- Software maintenance
- Assurance of the quality of the subcontractors' work and the customer-supplied parts.

4.3.1 Reviews

The design phase of the development process produces a variety of documents. The printed products include design reports, software test documents, software installation plans and software manuals, among others. Reviews can be categorized as formal design reviews (DRs) and peer reviews.

Formal design reviews (DRs)

A significant portion of these documents requires formal professional approval of their quality as stipulated in the development contract and demanded by the procedures applied by the software developer. It should be emphasized that the developer can continue to the next phase of the development process only on receipt of formal approval of these documents.

Ad hoc committees whose members examine the documents presented by the development teams usually carry out formal design reviews (widely known as "DRs"). The committees are composed of senior professionals, including the project leader and, usually, the department manager, the chief software engineer, and heads of other related departments. The majority of participants hold professional and administrative ranks higher than the project leader. On many occasions, the customer's representative will participate in a DR (this participation is generally indicated among the contractual arrangements).

The DR report itself includes a list of required corrections (termed "action items"). When a design review committee sits in order to decide upon the continuation of the work completed so far, one of the following options is usually open for consideration:

- Immediate approval of the DR document and continuation to the next development phase.
- Approval to proceed to the next development phase after all the action items have been completed and inspected by the committee's representative.
- An additional DR is required and scheduled to take place after all the action items have been completed and inspected by the committee's representative.

Peer reviews

Peer reviews (inspections and walkthroughs) are directed at reviewing short documents, chapters or parts of a report, a coded printout of a software module, and the like. Inspections and walkthroughs can take several forms and use many methods; usually, the reviewers are all peers, not superiors, who provide professional assistance to colleagues. The main objective of inspections and walkthroughs is to detect as many design and programming faults as possible. The output is a list of detected faults and, for inspections, also a defect summary and statistics to be used as a database for reviewing and improving development methods.

Because a peer's participation is usually voluntarily and viewed as a supplement to the regular workload, "reciprocity" considerations frequently enter. Thus, a current participant is expected to initiate a future inspection or walkthrough in which other colleagues will probably exchange roles regarding the inspection activities.

4.3.2 Expert opinions

Expert opinions support quality assessment efforts by introducing additional external capabilities into the organization's in-house development process. Turning to outside experts may be particularly useful in the following situations:

- Insufficient in-house professional capabilities in a given area.
- In small organizations in many cases it is difficult to find enough suitable candidates to participate in the design review teams. In such situations, outside experts may join a DR committee or, alternatively, their expert opinions may replace a DR.

- In small organizations or in situations characterized by extreme work pressures, an outside expert's opinion can replace an inspection.
- Temporary inaccessibility of in-house professionals (waiting will cause substantial delays in the project completion schedule).
- In cases of major disagreement among the organization's senior professionals, an outside expert may support a decision.

4.3.3 Software testing

Software tests are formal SQA components that are targeted toward review of the actual running of the software. The tests are based on a prepared list of test cases that represent a variety of expected scenarios. Software tests examine software modules, software integration, or entire software packages (systems). Recurrent tests (usually termed "regression tests"), carried out after correction of previous test findings, are continued till satisfactory results are obtained. The direct objective of the software tests, other than detection of software faults and other failures to fill the requirements, is the formal approval of a module or integration setup so that either the next programming phase can be begun or the completed software system can be delivered and installed.

Software testing programs are constructed from a variety of tests, some manual and some automated. All tests have to be designed, planned and approved according to development procedures. The test report will include a detailed list of the faults detected and recommendations about the performance of partial or complete recurrent tests following a subsequent round of corrections based on the test findings. (The advantages and disadvantages of automated testing are discussed later.) It is recommended that software tests be carried out by an independent, outside testing unit rather than by the project team, as the project team will naturally find it difficult to detect faults they failed to detect during development as well as to avoid conflicts of interest.

4.3.4 Software maintenance components

Software maintenance services vary in range and are provided for extensive periods, often several years. These services fall into the following categories:

- **Corrective maintenance** – User's support services and correction of software code and documentation failures.
- **Adaptive maintenance** – Adaptation of current software to new circumstances and customers without changing the basic software product. These adaptations are usually required when the hardware system or its components undergo modification (additions or changes).
- **Functionality improvement maintenance** – The functional and performance-related improvement of existing software, carried out with respect to limited issues.

Software maintenance services should meet all kinds of quality requirements, particularly functionality and scheduling requirements (generally decided together with the customer) as well as budget limitations (determined by the service provider). The provision of ongoing maintenance services involves the application of a great variety of SQA components. The main SQA components employed in the quality assurance of the maintenance system are as follows.

Pre-maintenance components

- Maintenance contract review
- Maintenance plan.

Software development life cycle components

These components are applied for functionality improvement and adaptive maintenance tasks, activities whose characteristics are similar to those of the software development process.

Infrastructure SQA components

- Maintenance procedures and instructions
- Supporting quality devices
- Maintenance staff training, retraining, and certification
- Maintenance preventive and corrective actions
- Configuration management
- Control of maintenance documentation and quality records.

Managerial control SQA components

- Maintenance service control
- Maintenance quality metrics
- Maintenance quality costs.

The above corresponding SQA components for the software development process have been described briefly in other sections of this overview. We will return to them in greater detail in the chapters dedicated to the individual topics.

4.3.5 Assurance of the quality of the external participant's work

Subcontractors and customers frequently join the directly contracted developers (the "supplier") in carrying out software development projects. The larger and more complex the project, the greater the likelihood that external participants will be required, and the larger the proportion of work transmitted to them (subcontractors, suppliers of COTS software and the customer). The motivation for turning to external participants lies in any number of factors, ranging from the economic to the technical to personnel-related interests, and reflects a growing trend in the allocation of the work involved with completing complex projects. The contribution of external participants may therefore vary. The assignment may thus concern carrying out phased tasks such as programming or testing, or the entire range of tasks required by a development stage of the project.

Most of the SQA controls applied to external participants are defined in the contracts signed between the relevant parties. If an external participant's work is performed using software assurance standards below those of the supplier's, risks of not meeting schedule or other requirements are introduced into the project. Hence, special software assurance efforts are required to establish effective controls over the external participant's work. Special SQA efforts are needed to assure the quality of the hardware, software, staff and training supplied by the customer.

4.4 Infrastructure components for error prevention and improvement

The goals of SQA infrastructure are the prevention of software faults or, at least, the lowering of software fault rates, together with the improvement of productivity. SQA infrastructure components are developed specifically to this end. These components are devised to serve a wide range of projects and software maintenance services. During recent years, we have witnessed the growing use of computerized automatic tools for the application of these components. This class of SQA components includes:

- Procedures and work instructions
- Templates and checklists
- Staff training, retraining, and certification
- Preventive and corrective actions
- Configuration management
- Documentation control.

4.4.1 Procedures and work instructions

Quality assurance procedures usually provide detailed definitions for the performance of specific types of development activities in a way that assures effective achievement of quality results. Procedures are planned to be generally applicable and to serve the entire organization. Work instructions, in contrast, provide detailed directions for the use of methods that are applied in unique instances and employed by specialized teams.

Procedures and work instructions are based on the organization's accumulated experience and knowledge; as such, they contribute to the correct and effective performance of established technologies and routines. Because they reflect the organization's past experience, constant care should be taken to update and adjust those procedures and instructions to current technological, organizational, and other conditions.

4.4.2 Supporting quality devices

One way to combine higher quality with higher efficiency is to use supporting quality devices, such as templates and checklists. These devices, based as they are on the accumulated knowledge and experience of the organization's development and maintenance professionals, contribute to meeting SQA goals by:

- Saving the time required to define the structure of the various documents or prepare lists of subjects to be reviewed.
- Contributing to the completeness of the documents and reviews.
- Improving communication between development team and review committee members by standardizing documents and agendas.

4.4.3 Staff training, instruction and certification

The banality of the statement that a trained and well-instructed professional staff is the key to efficient, quality performance, does not make this observation any less true. Within the framework of SQA, keeping an organization's human resources knowledgeable and updated at the level required is achieved mainly by:

- Training new employees and retraining those employees who have changed assignments.
- Continuously updating staff with respect to professional developments and the in-house, hands-on experience acquired.
- Certifying employees after their knowledge and ability have been demonstrated.

4.4.4 Preventive and corrective actions

Systematic study of the data collected regarding instances of failure and success contributes to the quality assurance process in many ways. Among them we can list:

- Implementation of changes that prevent similar failures in the future.
- Correction of similar faults found in other projects and among the activities performed by other teams.
- Implementing proven successful methodologies to enhance the probability of repeat successes.

The sources of these data, to mention only a few, are design review reports, software test reports, and customers' complaints. It should be stressed, however, that for these data to make a substantial contribution to quality, they must be systematically collected and professionally analyzed.

4.4.5 Configuration management

The regular software development and maintenance operations involve intensive activities that modify software to create new versions and releases. These activities are conducted throughout the entire software service period (usually lasting several years) in order to cope with the needed corrections, adaptations to specific customer requirements, application improvements, and so forth. Different team members carry out these activities simultaneously, although they may take place at different sites. As a result, serious dangers arise, whether of misidentification of the versions or releases, loss of the records delineating the changes implemented, or loss of documentation. Consequently failures may be caused.

Configuration management deals with these hazards by introducing procedures to control the change process. These procedures relate to the approval of changes, the recording of those changes performed, the issuing of new software versions and releases, the recording of the version and release specifications of the software installed in each site, and the prevention of any changes in approved versions and releases once they are issued. Most configuration management systems implement computerized tools to accomplish their tasks. These computerized systems provide the updated and proper versions of the installed software for purposes of further development or correction. Software configuration procedures generally authorize an administrator or a configuration management committee to manage all the required configuration management operations.

4.4.6 Documentation control

SQA requires the application of measures to ensure the efficient long-term availability of major documents related to software development ("controlled documents"). The purpose of one type of controlled document – the quality record – is mainly to provide evidence of the SQA system's performance. Documentation control therefore represents one of the building blocks of any SQA system.

Documentation control functions refer mainly to customer requirement documents, contract documents, design reports, project plans, development standards, etc. Documentation control activities entail:

- Definition of the types of controlled documents needed
- Specification of the formats, document identification methods, etc.
- Definition of review and approval processes for each controlled document
- Definition of the archive storage methods.

Controlled documents contain information important to the long-term development and maintenance of the software system, such as software test results, design review (DR) reports, problem reports, and audit reports. Quality records mainly contribute to the system's ability to respond to customer claims in the future.

4.5 Management SQA components

Managerial SQA components support the managerial control of software development projects and maintenance services. Control components include:

- Project progress control (including maintenance contract control)
- Software quality metrics
- Software quality costs.

4.5.1 Project progress control

The main objective of project progress control components is to detect the appearance of any situation that may induce deviations from the project's plans and maintenance service performance. Clearly, the effectiveness and efficiency of the corrective measures implemented is dependent on the timely discovery of undesirable situations.

Project control activities focus on:

- Resource usage
- Schedules
- Risk management activities
- The budget.

4.5.2 Software quality metrics

Measurement of the various aspects of software quality is considered to be an effective tool for the support of control activities and the initiation of process improvements during the development and the maintenance phases. These measurements apply to the functional quality, productivity, and organizational aspects of the project.

Among the software quality metrics available or still in the process of development, we can list metrics for:

- Quality of software development and maintenance activities
- Development teams' productivity
- Help desk and maintenance teams' productivity
- Software faults density
- Schedule deviations.

4.5.3 Software quality costs

The quality costs incurred by software development and application are, according to the extended quality costs model, the costs of control (prevention costs, appraisal costs, and managerial preparation and control costs) combined with the costs of failure (internal failure costs, external failure costs, and managerial failure costs). Management is especially interested in the total sum of the quality costs. It is believed that up to a certain level, expanding the resources allocated to control activities yields much larger savings in failure costs while reducing total quality costs. Accordingly, management tends to exhibit greater readiness to allocate funds to profitable proposals to improve application of existing SQA system components and further development of new components.

With respect to the specific SQA strategy applied, analysis of software quality costs can direct SQA efforts to the improvement of activities that cause significant failures with their attendant high failure costs or, alternatively, to make expensive control activities more efficient. This analysis, by directing attention to the teams whose activities keep their quality costs substantially below the average, enables others to learn from them and reproduce their success. Concomitantly, quality cost analysis can help identify those teams whose ineffective quality assurance efforts result in higher than average quality costs. The results can then be used to help the teams improve.

4.6 SQA standards, system certification, and assessment components

External tools offer another avenue for achieving the goals of software quality assurance. Specifically, the main objectives of this class of components are:

(1) Utilization of international professional knowledge.
(2) Improvement of coordination with other organizations' quality systems.
(3) Objective professional evaluation and measurement of the achievements of the organization's quality systems.

The standards available may be classified into two main sub-classes: quality management standards and project process standards. Either or both of the two sub-classes can be required by the customer and stipulated in the accompanying contractual agreements.

4.6.1 Quality management standards

The organization can clearly benefit from quality standards of the second sub-class that guide the management of software development, maintenance,

chapter 5

Contract review

Chapter outline

A bad contract is always an undesirable event. From the viewpoint of SQA, a bad contract – usually characterized by loosely defined requirements, and unrealistic budgets and schedules – is expected to yield low-quality software. So, it is natural for an SQA program to begin its preventive quality assurance efforts with a review of the proposal draft and, later, the contract draft ("contract review" covers both activities). The two reviews are aimed at improving the budget and timetable that provide the basis for the proposal and the subsequent contract, and revealing potential pitfalls at an early enough stage (in the proposal draft and in the contract draft).

This chapter is dedicated to the study of the objectives of contract review and the wide range of review subjects that correspond to these objectives. The contract review process originates in the customer–supplier relationship, and is expected to make a substantial contribution to internal projects as well.

After completing this chapter, you will be able to:

- Explain the two contract review stages.
- List the objectives of each stage of the contract review.
- Identify the factors that affect the extent of the review.
- Identify the difficulties in performing a major contract review.
- Explain the recommended avenues for implementing a major contract review.
- Discuss the importance of carrying out a contract review for internal projects.

5.1 Introduction: the CFV Project completion celebration

A happy gathering of the CFV project team at a popular restaurant downtown was called to celebrate the successful completion of a 10-month project for Carnegie Fruits and Vegetables, a produce wholesaler. The new information system registers product receipts from growers, processes clients' orders and produce shipments to clients (greengrocers and supermarkets), bills clients, and calculates payments made to the growers.

The team members were proud to emphasize that the project was conducted in full as originally scheduled. The team was especially jubilant as earlier that morning each member had received a nice bonus for finishing on time.

The third speaker, the software company's Vice President for Finance, altered the pleasant atmosphere by mentioning that this very successful project had actually lost about $90 000. During his remarks, he praised the planners for their good estimates of the resources needed for the analysis and design phase, and for the plans for broad reuse of software from other systems that were, this time, completely realized. "The only phase where our estimates failed was one of the project's final phases, the client's instruction, that where the client's staff are instructed on how to use the new information system. It now appears that no one had read the relevant RFP (requirement for proposal) section carefully enough. This section stated in a rather innocuous manner that the personnel in all the CFV branches where the software was to be installed would be instructed in its use by the software supplier." After a short pause he continued thus: "Nobody tried to find out how many branches our client operates. Nobody mentioned that CFV operates 19 branches – six of them overseas – before signing the contract!" He continued: "We tried to renegotiate the installation and instruction budget items with the client, but the client insisted on sticking to the original contract." Though no names were mentioned, it was clear that he blamed the sales negotiating team for the loss.

Similar, and in many cases much heavier, losses stem from sloppily written proposals or poorly understood contracts. Shallow and quick resource estimates, as well as exaggerated software sales efforts, have led to unrealistic schedules and budgets, or to unrealistic professional commitments. A proposal suffering from one of these faults or, worse, a combination of them and that later becomes a contract provides a certain recipe for project or service failure. It is clear that unrealistic professional commitments lead to failure to achieve the required software quality. Furthermore, in most cases, schedule and budget failures are accompanied by lower than acceptable software quality, due to pressures exerted on team members by management "to save time" and "to save resources". We can quite unrestrictedly state that such excessive pressures eventually lead to high rates of software failure.

Contract review is the software quality element that reduces the probability of such undesirable situations. Contract review is a requirement by the ISO 9001 standard and ISO 9000-3 Guidelines (see Sec. 4.3 of ISO (1997) and Sec. 7.2 of ISO/IEC (2001)). See Oskarsson and Glass (1996) for a discussion of some application aspects of contract review.

5.2 The contract review process and its stages

Several situations can lead a software company ("the supplier") to sign a contract with a customer. The most common are:

(1) Participation in a tender.
(2) Submission of a proposal according to the customer's RFP.
(3) Receipt of an order from a company's customer.
(4) Receipt of an internal request or order from another department in the organization.

Contract review is the SQA component devised to guide review drafts of proposal and contract documents. If applicable, contract review also provides oversight of the contacts carried out with potential project partners and subcontractors. The review process itself is conducted in two stages:

- **Stage One – Review of the proposal draft prior to submission to the potential customer** ("proposal draft review"). This stage reviews the final proposal draft and the proposal's foundations: customer's requirement documents, customer's additional details and explanations of the requirements, cost and resources estimates, existing contracts or contract drafts of the supplier with partners and subcontractors.
- **Stage Two – Review of contract draft prior to signing** ("contract draft review"). This stage reviews the contract draft on the basis of the proposal and the understandings (including changes) reached during the contract negotiations sessions.

The processes of review can begin once the relevant draft document has been completed. The individuals who perform the review thoroughly examine the draft while referring to a comprehensive range of review subjects. A checklist is very helpful for assuring the full coverage of relevant subjects (see Appendices 5A and 5B).

After the completion of a review stage it is required that the necessary changes, additions and corrections are introduced by the proposal team (after the proposal draft review) and by the legal department (after the contract draft review).

5.3 Contract review objectives

As can be expected, the two contract review stages have different objectives, which we detail in the following.

5.3.1 Proposal draft review objectives

The objective of the proposal draft review is to make sure that the following activities have been satisfactorily carried out.

(1) **Customer requirements have been clarified and documented.**

RFP documents and similar technical documents can be too general and imprecise for the project's purposes. As a result, additional details should be obtained from the customer. Clarifications of vague requirements and their updates should be recorded in a separate document that is approved by both the customer and the software firm.

(2) **Alternative approaches for carrying out the project have been examined.**

Often, promising and suitable alternatives on which to present a proposal have not been adequately reviewed (if at all) by the proposal team. This stipulation refers especially to alternatives encompassing software reuse, and partnerships or subcontracting with firms that have specialized knowledge or staff that can qualify for meeting the proposal's terms.

(3) **Formal aspects of the relationship between the customer and the software firm have been specified.**

The proposal should define formalities that include:

- Customer communication and interface channels
- Project deliverables and acceptance criteria
- Formal phase approval process
- Customer design and test follow-up method
- Customer change request procedure.

(4) **Identification of development risks.**

Development risks, such as insufficient professional know-how regarding the project's professional area or the use of required development tools, need to be identified and resolved. For a comprehensive description of identification of software risk items and methods for risk management actions, see Appendix 6A.

(5) **Adequate estimation of project resources and timetable.**

Resources estimation refers to professional staff as well as the project's budget, including subcontractors' fees. Scheduling estimates should take into account the time requirements of all the parties participating in the project.

Implementation tip

In some situations, a supplier deliberately offers a *below*-cost proposal, considering factors such as sales potential. In these cases, where the proposal is based on realistic estimates of schedule, budget and professional capabilities, the loss incurred is considered to be a calculated loss, not a contract failure.

(6) **Examination of the company's capacity with respect to the project.**
This examination should consider professional competence as well as the availability of the required team members and development facilities on the scheduled time.

(7) **Examination of the customer's capacity to meet his commitments.**
This examination refers to the customer's financial and organizational capacities, such as personnel recruitment and training, installation of the required hardware, and upgrading of its communications equipment.

(8) **Definition of partner and subcontractor participation.**
This covers quality assurance issues, payment schedules, distribution of project income/profits, and cooperation between project management and teams.

(9) **Definition and protection of proprietary rights.**
This factor is of vital importance in cases where reused software is inserted into a new package or when rights for future reuse of the current software need to be decided. This item also refers to the use of proprietary files of data crucial for operating the system and security measures.

The objectives of a proposal draft review are summarized in Frame 5.1.

Frame 5.1 Proposal draft review objectives

The nine proposal draft review objectives that make sure the following activities have been satisfactorily carried out:

1. Customer requirements have been clarified and documented.
2. Alternative approaches for carrying out the project have been examined.
3. Formal aspects of the relationship between the customer and the software firm have been specified.
4. Identification of development risks.
5. Adequate estimation of project resources and timetable have been prepared.
6. Examination of the firm's capacity with respect to the project.
7. Examination of the customer's capacity to fulfill his commitments.
8. Definition of partner and subcontractor participation conditions.
9. Definition and protection of proprietary rights.

5.3.2 Contract draft review objectives

The objectives of the contract draft review are to make sure that the following activities have been performed satisfactorily:

(1) No unclarified issues remain in the contract draft.
(2) All the understandings reached between the customer and the firm are to be fully and correctly documented in the contract and its appendices. These understandings are meant to resolve all the unclarified issues and differences between the customer and the firm that have been revealed so far.
(3) No changes, additions, or omissions that have not been discussed and agreed upon should be introduced into the contract draft. Any change, whether intentional or not, can result in substantial additional and unanticipated commitments on the part of the supplier.

The objectives of a contract draft review are summarized in Frame 5.2.

Frame 5.2 **Contract draft review objectives**

The three contract draft review objectives that make sure the following activities have been satisfactorily carried out:

1. No unclarified issues remain in the contract draft.
2. All understandings reached subsequent to the proposal are correctly documented.
3. No “new” changes, additions, or omissions have entered the contract draft.

5.4 Implementation of a contract review

Contract reviews vary in their magnitude, depending on the characteristics of the proposed project. This complexity may be either technical or organizational. Accordingly, different levels of professional effort are justified for the various contract reviews. Special professional efforts are required for major proposals.

5.4.1 Factors affecting the extent of a contract review

The most important project factors determining the extent of the contract review efforts required are:

- **Project magnitude**, usually measured in man-month resources.
- **Project technical complexity.**

- **Degree of staff acquaintance with and experience in the project area.** Acquaintance with the project area is frequently linked with software reuse possibilities; in cases where a high proportion of software reuse is possible, the extent of the review is reduced.
- **Project organizational complexity.** The greater the number of organizations (i.e., partners, subcontractors, and customers) taking part in the project, the greater the contract review efforts required.

We may therefore assume that "simple" contract reviews will be carried out by one reviewer, who will focus on a few subjects and invest little time in his review. However, a large-scale contract review may require the participation of a team to examine a wide range of subjects, a process demanding the investment of many working hours.

5.4.2 Who performs a contract review?

The task of contract review can be completed by various individuals, listed here in ascending order, according to the complexity of the project:

- The leader or another member of the proposal team.
- The members of the proposal team.
- An outside professional or a company staff member who is not a member of the proposal team.
- A team of outside experts. Usually, a contract review team composed of outside experts is called in, especially for major proposals (see Section 5.4.3). Outside experts may be called also for contract reviews in small software development organizations that are unable to find enough adequate team members in their staff.

5.4.3 Implementation of a contract review for a major proposal

Major proposals are proposals for projects characterized by at least some of the following: very large-scale project, very high technical complexity, new professional area for the company, and high organizational complexity (realized by a great number of organizations, i.e., partners, subcontractors, and customers, that take part in the project). Implementation of a contract review process for a major project usually involves substantial organizational difficulties. Some avenues for overcoming these difficulties are suggested here, following a review of the factors that introduce difficulties to a smooth completion of the task.

The difficulties of carrying out contract reviews for major proposals

Almost everybody agrees that contract review is a major procedure for reducing the risks of major project failures. Several substantial, fundamental, and inherent difficulties in performing the contract review exist, especially for those situations requiring a review of a major proposal.

- **Time pressures.** Both stages of the contract review, proposal draft review and contract draft review are usually performed when the tender team is under considerable time pressures. As a result, each stage of the contract review has to be completed within a few days to allow for the subsequent corrections of documents to take place.
- **Proper contract review requires substantial professional work.** Professional performance of each stage of the contract review requires investment of substantial professional expertise (the amount of time required varies, of course, according to the nature of the project).
- **The potential contract review team members are very busy.** The potential members of the contract review team are often senior staff members and experts who usually are committed to performing their regular tasks at the very time that the review is needed. Freeing professional staff can therefore be a significant logistical problem.

Recommended avenues for implementing major contract reviews

The careful planning of contract reviews is required for their successful completion. As should be clear by now, this holds doubly for major contract reviews. It is recommended that the following steps be taken to facilitate the review process.

- **The contract review should be scheduled.** Contract review activities should be included in the proposal preparation schedule, leaving sufficient time for the review and the ensuing corrections to be made.
- **A team should carry out the contract review.** Teamwork makes it possible to distribute the workload among the team members so that each member of the contract review team can find sufficient time to do his or her share (which may include preparing a written report that summarizes his or her findings and recommendations).
- **A contract review team leader should be appointed.** It is important that the responsibility for organizing, managing and controlling the contract review activities be defined, preferable by appointing a team leader. The activities of the team leader include:
  - Recruitment of the team members
  - Distribution of review tasks among the team's members
  - Coordination between the members of the review team
  - Coordination between the review team and the proposal team
  - Follow-up of activities, especially compliance with the schedule
  - Summarization of the findings and their delivery to the proposal team.

Implementation tip

As contract reviews may impose a substantial workload and additional pressures on the proposal team, thought should be given to when it may be appropriate to abstain from conducting a contract review. Such situations may occur with small-scale projects, or small- to medium-scale cost-plus projects. Contract review procedures should therefore define those types of projects for which a contract review is not obligatory.

For other defined types of "simple" projects, it is recommended that authority be given to a senior manager to make the decision as to whether to perform the review.

5.5 Contract review subjects

Contract reviews examine many subjects, based on the contract review objectives. Checklists are useful devices for helping review teams to organize their work and achieve high coverage of the relevant subjects. It is clear that many of the subjects on these lists are irrelevant for any specific project. At the same time, even a comprehensive checklist may exclude some important subjects relevant to a given project proposal. It is the task of the contract review team, but especially of its leader, to determine the list of subjects pertinent for the specific project proposal.

Lists of contract review subjects, classified according to contract review objectives, are presented in the appendices to this chapter:

- Appendix 5A: Proposal draft review – subjects checklist
- Appendix 5B: Contract draft review – subjects checklist.

5.6 Contract reviews for internal projects

A substantial number, if not the majority, of software projects are internal projects — "in-house" projects – carried out by one unit of an organization for another unit of the same organization. In such cases, the software development unit is the supplier, while the other unit can be considered the customer. Typical internal projects and their in-house customers are listed in Table 5.1.

Frequently, internal software development projects are not based on what would be considered a complete customer–supplier relationship. In many cases, these projects are based on general agreements, with goodwill playing an important role in the relationships between the two units. It follows that the developing unit will perform only a short and "mild" contract review, or none at all.

chapter 6

Development and quality plans

Chapter outline

Imagine that you have just been appointed head of a sizable project. As is often the case in the software industry, you come under serious time pressures from the very first day. Because you were a member of the proposal team and participated in most of the meetings held with the customer's representatives, you are confident that you know all that is necessary to do the job. You intend to use the proposal plans and internal documents that the team had prepared as your development and quality plans. You are prepared to rely on these materials because you know that the proposal and its estimates, including the timetable, staff requirements, list of project documents, scheduled design reviews, and list of development risks have all been thoroughly reviewed by the contract review team.

You are therefore a bit disappointed that at this crucial point of the project, the Development Department Manager demands that you immediately prepare new and separate project development plans ("development plan") and project quality plans ("quality plan"). When you claim that the completed proposal and its appendices could serve as the requested plans, the manager insists that they be updated, with new and more comprehensive topics added to guarantee the plans' adequacy. "By the way," the manager mentions almost as an aside, "don't forget that a period of seven months has elapsed between the proposal preparation and the final signing of the contract. Such a period is a hell of time in our trade . . . ."

You should expect that your department manager is right. The effort invested in preparing the development and quality plans will certainly be beneficial. You may discover that some team members will not be available at the scheduled dates due to delays in completion of their current assignments, or that the consulting company that had agreed to provide professional support in a highly specialized and crucial area has suffered heavy losses and gone bankrupt in the interim. These are just two of the types of problems that can arise.

To sum up, the project needs development and quality plans that:

- Are based on proposal materials that have been re-examined and thoroughly updated.
- Are more comprehensive than the approved proposal, especially with respect to schedules, resource estimates, and development risk evaluations.
- Include additional subjects, absent from the approved proposal.
- Were prepared at the beginning of the project to sound alerts regarding scheduling difficulties, potential staff shortages, paucity of development facilities, problems with meeting contractual milestones, modified development risks, and so on.

Development and quality plans are major elements needed for project compliance with 9000.3 standards (see Sections 4.2 and 4.4 of ISO (1997) and Sections 7.1 and 7.3 of ISO/IEC (2001), and with the IEEE 730 standard (IEEE, 1998). It is also an important element in the Capability Maturity Model (CMM) for assessment of software development organization maturity (see Paulk *et al.*, 1995, Sec. 7.2; Humphrey, 1989; Felschow, 1999). Given their importance, these plans deserve a special chapter.

Therefore, this chapter is dedicated to the study of project development and quality plans, their objectives and elements.

After completing this chapter, you will be able to:

- Explain the objectives of a development plan and a quality plan.
- Identify the elements of a development plan.
- Identify the elements of a quality plan.

- Identify the major software risk items.
- Explain the process of software risk management.
- Discuss the importance of development and quality plans for small projects.
- Discuss the importance of development and quality plans for internal projects.

6.1 Development plan and quality plan objectives

Planning, as a process, has several objectives, each of which is meant to prepare adequate foundations for the following:

(1) Scheduling development activities that will lead to the successful and timely completion of the project, and estimating the required manpower resources and budget.
(2) Recruiting team members and allocating development resources (according to activity schedules and manpower resource requirement estimates).
(3) Resolving development risks.
(4) Implementing required SQA activities.
(5) Providing management with data needed for project control.

6.2 Elements of the development plan

Based on the proposal materials, the project's development plan is prepared to fulfill the above objectives. The following elements, each applicable to different project components, comprise a project development plan.

(1) **Project products**
The development plan includes the following products:
   - Design documents specifying dates of completion, indicating those items to be delivered to the customer ("deliverables")
   - Software products (specifying completion date and installation site)
   - Training tasks (specifying dates, participants and sites).

(2) **Project interfaces**
Project interfaces include:
   - Interfaces with existing software packages (software interface)
   - Interfaces with other software and/or hardware development teams that are working on the same system or project (i.e., cooperation and coordination links)
   - Interfaces with existing hardware (hardware interface).

(3) **Project methodology and development tools** to be applied at each phase of the project

Implementation tip

When evaluating the suitability of proposed project methodology and development tools, one should also take into account the professional experience of the staff, including the subcontractors' personnel, even if temporary.

(4) **Software development standards and procedures**
A list should be prepared of the software development standards and procedures to be applied in the project.

(5) **The mapping of the development process**
Mapping of the development process involves providing detailed definitions of each of the project's phases. These descriptions include definitions of inputs and outputs, and the specific activities planned. Activity descriptions include:

(a) An estimate of the activity's duration. These estimates are highly dependent on the experience gained in previous projects.

(b) The logical sequence in which each activity is to be performed, including a description of each activity's dependence on previously completed activities.

(c) The type of professional resources required and estimates of how much of these resources are necessary for each activity.

Implementation tip

SQA activities, such as design review and software tests, should be included among the scheduled project activities. The same applies to the design and code correction activities. Failing to schedule these activities can cause unanticipated delays in the initiation of subsequent activities.

Several methods are available for scheduling and graphically presenting the development process. One of the most commonly used methods is the GANTT chart, which displays the various activities by horizontal bars whose lengths are proportional to the activity's duration. The bars represent the activities themselves, and are placed vertically, according to their planned initiation and conclusion. Several computerized tools can prepare GANTT charts in addition to producing lists of activities by required time for their beginning and conclusion, and so forth.

More advanced scheduling methodologies, such as CPM and PERT, both of which belong to the category of critical path analysis, take sequence dependencies into account in addition to duration of activities. They enable calculation of the earliest and latest acceptable start times for each activity. The difference between start times determines the activity's scheduling flexibility. Special attention is awarded to those activities lacking scheduling flexibility (which explains their being called "critical path" activities), and whose tardy completion may cause delay in the conclusion of the entire project.

Several software packages, used in conjunction with these methodologies, support the planning, reporting and follow-up of project timetables. An example of a software package of this type is *Microsoft Project*™. For a more detailed discussion of scheduling, refer to the literature dealing with project management.

(6) **Project milestones**
For each milestone, its completion time and project products (documents and code) are to be defined.

(7) **Project staff organization**
The organization plan comprises:

- Organizational structure: definition of project teams and their tasks, including teams comprised of a subcontractor's temporary workers.
- Professional requirements: professional certification, experience in a specific programming language or development tool, experience with a specific software product and type, and so forth.
- Number of team members required for each period of time, according to the activities scheduled. It is expected that teams will commence their activities at different times, and that their team size may vary from one period to the next, depending on the planned activities.
- Names of team leaders and team members. Difficulties are expected to arise with respect to the long-term assignment of staff members to teams because of unanticipated changes in their current assignments. Therefore, the names of staff are required to help keep track of their participation as team members.

Implementation tip

The long-term availability of project staff should be carefully examined. Lags in completing former assignments may result in delays in joining the project team, which increases the risk of failing to meet project milestones. In addition, staff "evaporation" caused by resignations and/or promotions, phenomena that are particularly frequent in the software industry, can cause staff shortages. Therefore, estimates of staff availability should be examined periodically to avoid "surprises". Early warning of unforeseen staff shortages makes it easier to resolve the problem.

(8) **Development facilities**
Required development facilities include hardware, software and hardware development tools, office space, and other items. For each facility, the period required for its use should be indicated on the timetable.

(9) **Development risks**
Development risks are inherent in any project. To understand their pervasiveness, and how they can be controlled, we should first define the concept. A development risk is "a state or property of a development task or environment, which, if ignored, will increase the likelihood of project failure" (Ropponen and Lyytinen, 2000). Typical development risks are:

- Technological gaps – Lack of adequate and sufficient professional knowledge and experience to carry out the demands of the development contract.
- Staff shortages – Unanticipated shortfalls of professional staff.
- Interdependence of organizational elements – The likelihood that suppliers of specialized hardware or software subcontractors, for example, will not fulfill their obligations on schedule.

The top 10 major software risk items, as listed by Boehm and Ross (1989), are shown in the Appendix to this chapter in Table 6A.1. Systematic risk management activities should be initiated to deal with them. The risk management process includes the following activities: risk identification, risk evaluation, planning of risk management actions (RMAs), implementation of RMAs, and monitoring of RMAs. Software RMAs are incorporated in the development plan.

For further discussion of software development risks and software risk management, see Appendix 6A.

The growing importance of software risk management is expressed in the spiral model for software development. To cope with this type of risk, a special phase dedicated to software risk assessment is assigned to every cycle of the spiral (Boehm, 1988, 1998).

(10) **Control methods**
In order to control project implementation, the project manager and the department management apply a series of monitoring practices when preparing progress reports and coordinating meetings. A comprehensive discussion of project control methods is found in Chapter 19.

(11) **Project cost estimation**
Project cost estimates are based on proposal costs estimates, followed by a thorough review of their continued relevance based on updated human resource estimates, contracts negotiated with subcontractors and suppliers, and so forth. For instance, part of the project, planned to be carried out by an internal development team, needs to be performed by a subcontractor, due to unavailability of the team. A change of this nature is usually involved in a substantial additional budget.

The elements comprising a development plan are listed in Frame 6.1.

Frame 6.1 The elements comprising a development plan

1. Project products, specifying "deliverables"
2. Project interfaces
3. Project methodology and development tools
4. Software development standards and procedures
5. Map of the development process
6. Project milestones
7. Project staff organization
8. Required development facilities
9. Development risks and risk management actions
10. Control methods
11. Project cost estimates

Development plan approval

Development plan review and approval is to be completed according to the procedures applied within the organization.

6.3 Elements of the quality plan

All or some of the following items, depending on the project, comprise the elements of a project quality plan:

(1) **Quality goals**

The term "quality goals" refers to the developed software system's substantive quality requirements. Quantitative measures are usually preferred to qualitative measures when choosing quality goals because they provide the developer with more objective assessments of software performance during the development process and system testing. However, one type of goal is not totally equivalent to the other. The possible replacement of qualitative with quantitative measures is illustrated in the following example.

Example

A software system to serve the help desk operations of an electrical appliance manufacturer is to be developed. The help desk system (HDS) is intended to operate for 100 hours per week. The software quality assurance team was requested to prepare a list of quantitative quality goals appropriate to certain qualitative requirements, as shown in Table 6.1.

Table 6.1: Help desk requirements and quantitative goals

HDS qualitative requirements	Related quantitative quality goals
The HDS should be user friendly	A new help desk operator should be able to learn the details of the HDS following a course lasting less than 8 hours, and to master operation of the HDS in less than 5 working days.
The HDS should be very reliable	HDS availability should exceed 99.5% (HDS downtime should not exceed 30 minutes per week).
The HDS should operate continuously	The system's recovery time should not exceed 10 minutes in 99% of cases of HDS failure.
The HDS should be highly efficient	An HDS operator should be able to handle at least 100 customer calls per 8-hour shift.
The HDS should provide high quality service to the applying customers	Waiting time for an operator response should not exceed 30 seconds in 99% of the calls. Achievement of this goal depends on the combination of software features and number of workstations installed and operated.

The quality goals should reflect the major acceptance criteria indicated in the customer's requirement document (i.e., the RFP document). As such, quality goals serve as measures of the successful achievement of the customer's quality requirements.

(2) **Planned review activities**

The quality plan should provide a complete listing of all planned review activities: design reviews (DRs), design inspections, code inspections, and so on, with the following determined for each activity:

- The scope of the review activity
- The type of the review activity
- The schedule of review activities (as defined by its priority and the succeeding activities of the project process)
- The specific procedures to be applied
- Who is responsible for carrying out the review activity?

(3) **Planned software tests**

The quality plan should provide a complete list of planned software tests, with the following designated for each test:

- The unit, integration or the complete system to be tested
- The type of testing activities to be carried out, including specification of computerized software tests to be applied
- The planned test schedule (as defined by its priority and the succeeding activities of the project process)

- The specific procedures to be applied
- Who is responsible for carrying out the test.

(4) **Planned acceptance tests for externally developed software**

A complete list of the acceptance tests planned for externally developed software should be provided within the quality plan. Items to be included are (a) purchased software, (b) software developed by subcontractors, and (c) customer-supplied software. The acceptance tests for externally developed software should parallel those used for internally developed software tests.

(5) **Configuration management**

The quality plan should specify configuration management tools and procedures, including those change-control procedures meant to be applied throughout the project.

The required software quality plan elements are listed in Frame 6.2.

Frame 6.2 Elements of a software quality plan

1. List of quality goals
2. Review activities
3. Software tests
4. Acceptance tests for software externally developed
5. Configuration management tools and procedures

The quality plan document, its format and approval

The quality plan may be prepared as part of the development plan or as an independent document. In some cases, the plan is divided into several documents by item category, such as DR plan, testing plan, and plan for externally developed software acceptance tests. Review and approval of the quality plan should be conducted according to the organization's standard procedures for such plans.

6.4 Development and quality plans for small projects and for internal projects

It is quite natural for project leaders to try to evade the "hassle" of preparing a development plan and a quality plan (and the hustle surrounding reviews and plan approvals). This behavior reflects the tendency to avoid "bureaucracy work" and the sweeping control that customers may attempt to exercise. This tendency is especially common in two different situations: small projects and internal projects. The argument for preparing these plans for such projects is discussed in the following two sections.

6.4.1 Development plans and quality plans for small projects

- Does a project of only 40 working days' duration, to be performed by one professional and completed within 12 weeks, justify the investment of a man-day in order to prepare full-scale development and quality plans?
- Does a project to be implemented by three professionals, with a total investment of 30 man-days and completed within five weeks, require full-scale plans?

It should be clear that the development and quality plan procedures applicable to large projects cannot be automatically applied to small projects. Special procedures are needed. These procedures determine how to treat the project in question with respect to the plans:

(1) Cases/situations where neither development nor quality plans are required, e.g. projects requiring 15 man-days.
(2) Cases/situations where the decision to prepare the plans is left to the project leader's discretion. One example could be a project requiring less than 50 man-days where no significant software risk item had been identified – in this case it might be decided that no plans will be prepared. Another example could be a small but complicated project that has to be completed within 30 days, in which there is a heavy penalty on not being completed on time. In this case, planning is needed in order to cope with the project difficulties.
(3) Cases/situations where development and quality plans are obligatory.

A list of elements recommended for inclusion in development and quality plans for small projects is shown in Frame 6.3.

Frame 6.3 Recommended elements of development and quality plans for small projects

The development plan:

- Project products, indicating "deliverables"
- Project benchmarks
- Development risks
- Estimates of project costs

The quality plan:

- Quality goals

Several advantages to "planned" small projects over "unplanned" projects can be identified, even for "reduced" plans:

(1) A more comprehensive and thorough understanding of the task is attained.
(2) Greater responsibility for meeting obligations can be assigned.
(3) It becomes easier for management and customers to share control of the project and to identify unexpected delays early on.
(4) Better understandings with respect to the requirements and timetable can be reached between the developer and the customer.

6.4.2 Development plans and quality plans for internal projects

Internal projects are those projects intended for use by other departments in the organization or by the entire organization, as well as those projects dealing with software package development for the software market. Common to all these project types is the fact that no external body participates as customer in their development. Internal projects can be of medium or large scale. Yet even in these cases, there is a tendency to avoid preparation of adequate development and quality plans. The following example illustrates the negative consequences of an "unplanned" internal project.

Example
The Marketing Department of Toyware Ltd, a new computer games manufacturer, had planned to hit the market with "Super-Monster 2000", the firm's new, advanced computer game, during the upcoming Christmas season. The Software Development Department claimed that work on the game should commence immediately in order to complete the project on time. Therefore, preparation of a proposal for discussion by the Marketing and Software Development Departments, and the subsequent preparation of development and quality plans, were not viewed as necessary. The Development Department estimated the project budget at $240 000, which was transferred to the Department. According to the marketing timetable, system tests were to be completed no later than 1 October so as to allow the Marketing Department to carry out the required promotion and advertising campaigns in time for the Christmas sales season.

As the project progressed, it appeared that there might be a delay, but only at the end of June was it obvious that a three-month delay could not be avoided. The promotional and advertising activities that had taken place before 30 June thus became worthless. The project was finally completed at the end of February. The project's cost overrun was significant – actual costs exceeded $385 000 – but most painful was the company's lost opportunity to exploit the Christmas market. Last week, the company's management decided to avoid any future internal computer game development projects.