part III

SQA components in the project life cycle

The project life cycle encompasses two stages: the development life cycle stage and the operation–maintenance stage. Most of the SQA components to be reviewed in Part III support at least one of the phases comprising these stages.

Development life cycle SQA components are meant to detect design and programming errors in the design and programming (coding) phases. The components applied in this stage belong to one of the following four sub-classes:

- Formal design reviews
- Peer reviews
- Expert opinions
- Software testing.

Operation–maintenance stage SQA components include special components to be applied for corrective maintenance but also development life cycle SQA components that can also be used for functionality improvement maintenance tasks.

An additional sub-class of SQA components, other than those listed above, deals with assuring the quality of project parts performed by subcontractors and other external participants during the project life cycle. The importance of this sub-class stems from the high risks associated with functional failures and the failure to keep to the project timetable. Both types of risk are directly related to the difficulty of controlling the external participants' performance.

The project characteristics determine which SQA components enter the project life cycle at any particular point. To guarantee their effectiveness, the choice of components is decided upon prior to the project's initiation.

The first chapter of this part, Chapter 7, is dedicated to a general discussion of the integration of software quality assurance components within each phase of the project's life cycle. A model for assessing the effectiveness and costs of integration is also presented in this chapter.

Chapter 8 discusses the review components of the design phase: formal design reviews, peer review and expert opinions.

Chapters 9 and 10 are dedicated to software testing issues, with Chapter 9 focusing on testing strategies and Chapter 10 on software testing

implementation. Among the implementation issues discussed are manual and automated testing.

Chapter 11 deals with SQA components appropriate to the operation–maintenance stage. Although functionality improvement and adaptive maintenance tasks employ primarily development life cycle SQA components (see Chapters 8–10), corrective maintenance, the subject of this chapter, has distinctive requirements and special SQA components.

Chapter 12, the final chapter in this part, explores the SQA issues raised by the participation of external participants in a project.

chapter 7

Integrating quality activities in the project life cycle

Chapter outline

The first part of this chapter is dedicated to the various software development models in current use. The remaining sections deal with the objectives of the software quality assurance activities conducted throughout the project life cycle, their integration in the development process, and the factors considered before applying them.

One might inquire why not begin with SQA activities and omit the discussion of the software development models? This question is not simply rhetorical. Software development models provide a coordinated set of concepts and methodologies needed to implement software development. As such, they include definitions of the main activities needed for development, the appropriate sequence for their performance, and their milestones. By deciding what models are to be applied, the project leader determines how the project will be carried out. Most quality assurance activities take place in conjunction with the completion or examination of activity milestones, which require review of the product development activities previously

completed. Therefore, SQA professionals should be acquainted with the various software engineering models in order to be able to prepare a quality plan that is properly integrated into the project plan.

The rest of the first part of the chapter deals with the factors affecting the choice of software quality activities to be integrated in the development process. The following four chapters (Chapters 8 to 11) deal with the specific software quality methodologies to be applied at each phase of the development stage and in the operation–maintenance stage.

The second part of the chapter is dedicated to a model for assessing a plan for SQA defect-removal effectiveness and cost. The model, a multiple filtering model, is based on data acquired from a survey of defect origins, percentages of defect removal achieved by various quality assurance activities, and the defect-removal costs incurred at the various development phases. The model enables quantitative comparison of quality assurance policies as realized in quality assurance plans.

After completing this chapter, you will be able to:

- Describe the various software development models and discuss the differences between them.
- Explain the considerations affecting intensity of applying quality assurance activities.
- Explain the different aspects of verification, validation and qualification associated with quality assurance activities.
- Describe the model for the SQA plan's defect-removal effectiveness and cost.
- Explain possible uses for the model.

7.1 Classic and other software development methodologies

Four models of the software development process are discussed in this section:

- The Software Development Life Cycle (SDLC) model
- The prototyping model
- The spiral model
- The object-oriented model.

The models presented here are not merely alternatives; rather, they represent complementary view of software development or refer to different development contexts.

The Software Development Life Cycle model (the SDLC model) is the classic model (still applicable today); it provides the most comprehensive description of the process available. The model displays the major building blocks for the entire development process, described as a linear sequence. In the initial phases of the software development process, product design documents

are prepared, with the first version of the computer program completed and presented for evaluation only at quite a late stage of the process. The SDLC model can serve as a framework within which the other models are presented.

The prototyping model is based on replacement of one or more SDLC model phases by an evolutionary process, where software prototypes are used for communication between the developer and the users and customers. Prototypes are submitted to user representatives for evaluation. The developer then continues development of a more advanced prototype, which is also submitted for evaluation. This evolutionary process continues till the software project is completed or the software prototype has reached the desired phase. In this case, the rest of the development process can be carried out according to a different methodology, for example the classic SDLC model.

The spiral model provides a methodology for ensuring effective performance at each of the SDLC model phases. It involves an iterative process that integrates customer comments and change requirements, risk analysis and resolution, and software system planning and engineering activities. One or more iterations of the spiral model may be required to complete each of the project's SDLC phases. The associated engineering tasks may be performed according to any one model or a combination of them.

The object-oriented model incorporates large-scale reuse of software by integrating reusable modules into new software systems. In cases where no reusable software modules (termed objects or components) are available, the developer may perform a prototyping or SDLC process to complete the newly developed software system.

All four models will be presented in detail in the next four sections. Detailed discussions of the respective methodologies are available in the software engineering and system analysis literature, particularly Pressman (2000) and Kendall and Kendall (1999).

7.1.1 The software development life cycle (SDLC) model

The classic Software Development Life Cycle (SDLC) model is a linear sequential model that begins with requirements definition and ends with regular system operation and maintenance. The most common illustration of the SDLC model is the waterfall model, shown in Figure 7.1.

The model shown in Figure 7.1 presents a seven-phase process, as follows:

- **Requirements definition.** For the functionality of the software system to be developed, the customers must define their requirements. In many cases the software system is part of a larger system. Information about the other parts of the expanded system helps establish cooperation between the teams and develop component interfaces.
- **Analysis.** The main effort here is to analyze the requirements' implications to form the initial software system model.

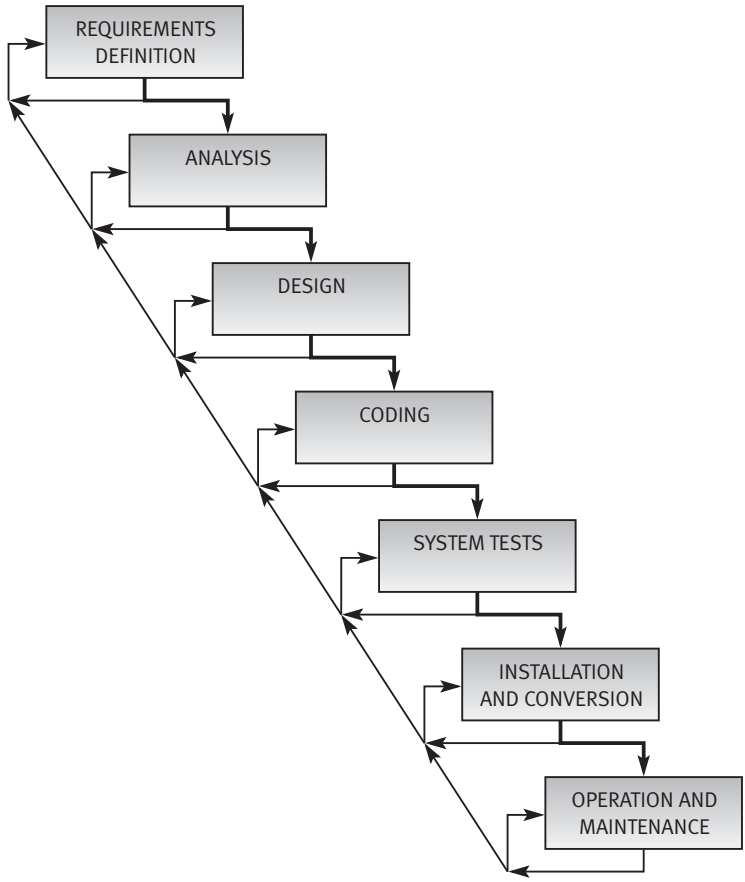


Figure 7.1: The waterfall model
Source: After Boehm (1981) and Royce (1970) (© 1970 IEEE)

- **Design.** This stage involves the detailed definition of the outputs, inputs and processing procedures, including data structures and databases, software structure, etc.
- **Coding.** In this phase, the design is translated into a code. Coding involves quality assurance activities such as inspection, unit tests and integration tests.
- **System tests.** System tests are performed once the coding phase is completed. The main goal of testing is to uncover as many software errors as possible so as to achieve an acceptable level of software quality once corrections have been completed. System tests are carried out by the software developer before the software is supplied to the customer. In many cases the customer performs independent software tests ("acceptance tests") to assure him or herself that the developer has fulfilled all the commitments and that no unanticipated or faulty software reactions are anticipated. It is quite common for a customer to ask the developer to

join him or her in performing joint system tests, a procedure that saves the time and resources required for separate acceptance tests.

- **Installation and conversion.** After the software system is approved, the system is installed to serve as firmware, that is, as part of the information system that represents a major component of the expanded system. If the new information system is to replace an existing system, a software conversion process has to be initiated to make sure that the organization's activities continue uninterrupted during the conversion phase.
- **Regular operation and maintenance.** Regular software operation begins once installation and conversion have been completed. Throughout the regular operation period, which usually lasts for several years or until a new software generation appears on the scene, maintenance is needed. Maintenance incorporates three types of services: corrective – repairing software faults identified by the user during operation; adaptive – using the existing software features to fulfill new requirements; and perfective – adding new minor features to improve software performance.

The number of phases can vary according to the characteristics of the project. In complex, large-scale models, some phases are split, causing their number to grow to eight, nine or more. In smaller projects, some phases may be merged, reducing the number of phases to six, five or even four phases.

At the end of each phase, the outputs are examined and evaluated by the developer and, in many cases, by the customer as well. Possible outcomes of the review and evaluation include:

- Approval of the phase outputs and progress on to the next phase, or
- Demands to correct, redo or change parts of the last phase; in certain cases, a return to earlier phases is required.

The width of the lines connecting the rectangular boxes in the illustration reflects the relative probabilities of the different outcomes. Thus, the most commonly performed process is a linear sequence (no or only minor corrections). We should note, however, that the model emphasizes direct development activities and does not indicate customer stakes in the development process.

The classic waterfall model was suggested by Royce (1970) and later presented in its commonly known form by Boehm (1981). It provides the foundations for the majority of the major software quality assurance standards employed, such as IEEE Std 1012 (IEEE, 1998) and IEEE Std 12207 (IEEE, 1996, 1997a, 1997b), to mention just two.

7.1.2 The prototyping model

The prototyping methodology makes use of (a) developments in information technology, namely, advanced application generators that allow for fast and easy development of software prototypes, combined with (b) active

participation in the development process by customers and users capable of examining and evaluating prototypes.

When applying the prototyping methodology, future users of the system are required to comment on the various versions of the software prototypes prepared by the developers. In response to customer and user comments, the developers correct the prototype and add parts to the system on the way to presenting the next generation of the software for user evaluation. This process is repeated till the prototyping goal is achieved or the software system is completed. A typical application of the prototyping methodology is shown in Figure 7.2.

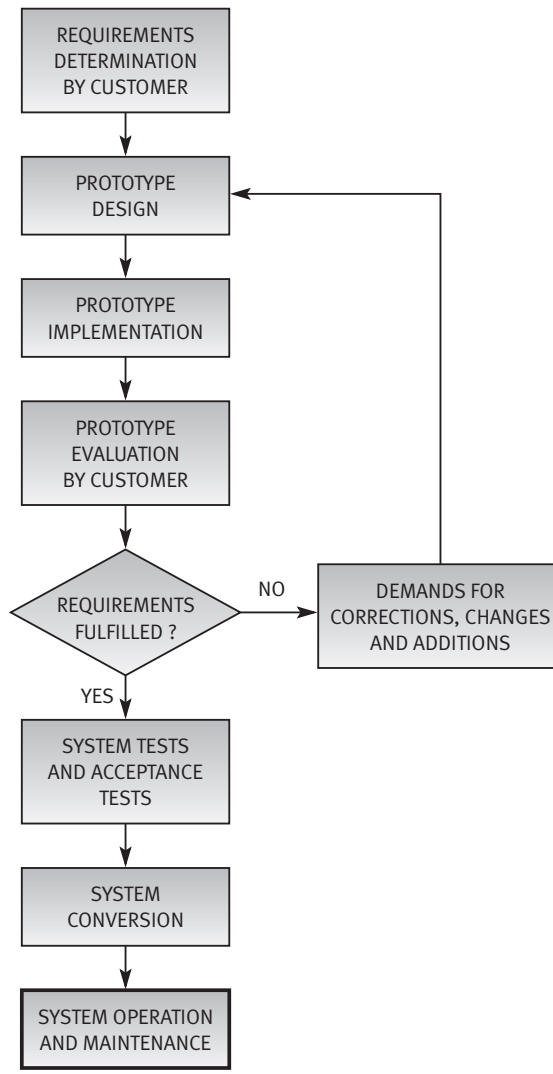


Figure 7.2: The prototyping model

Prototyping can be applied in combination with other methodologies or as a "stand alone" methodology. In other words, the extent of prototyping can vary, from replacement of one SDLC (or other methodology) phase up to complete prototyping of the entire software system.

Prototyping as a *software development* methodology has been found to be efficient and effective mainly for small- to medium-sized software development projects. The main advantages and deficiencies of prototyping over the complete SDLC methodology, summarized in Frame 7.1, result from the user's intense involvement in the software development process. Such involvement facilitates the user's understanding of the system while it limits the developer's freedom to introduce innovative changes in the system.

Frame 7.1 Prototyping versus SDLC methodology – advantages and disadvantages (mainly for small to medium-sized projects)

Advantages of prototyping:

- Shorter development process
- Substantial savings of development resources (man-days)
- Better fit to customer requirements and reduced risk of project failure
- Easier and faster user comprehension of the new system

Disadvantages of prototyping:

- Diminished flexibility and adaptability to changes and additions
- Reduced preparation for unexpected instances of failure

7.1.3 The spiral model

The spiral model, as revised by Boehm (1988, 1998), offers an improved methodology for overseeing large and more complex development projects displaying higher prospects for failure, typical of many projects begun in the last two decades. It combines an iterative model that introduces and emphasizes risk analysis and customer participation into the major elements of SDLC and prototyping methodologies.

According to the spiral model, shown in Figure 7.3, software development is perceived to be an iterative process; at each iteration, the following activities are performed:

- Planning
- Risk analysis and resolution
- Engineering activities according to the stage of the project: design, coding, testing, installation and release
- Customer evaluation, including comments, changes and additional requirements, etc.

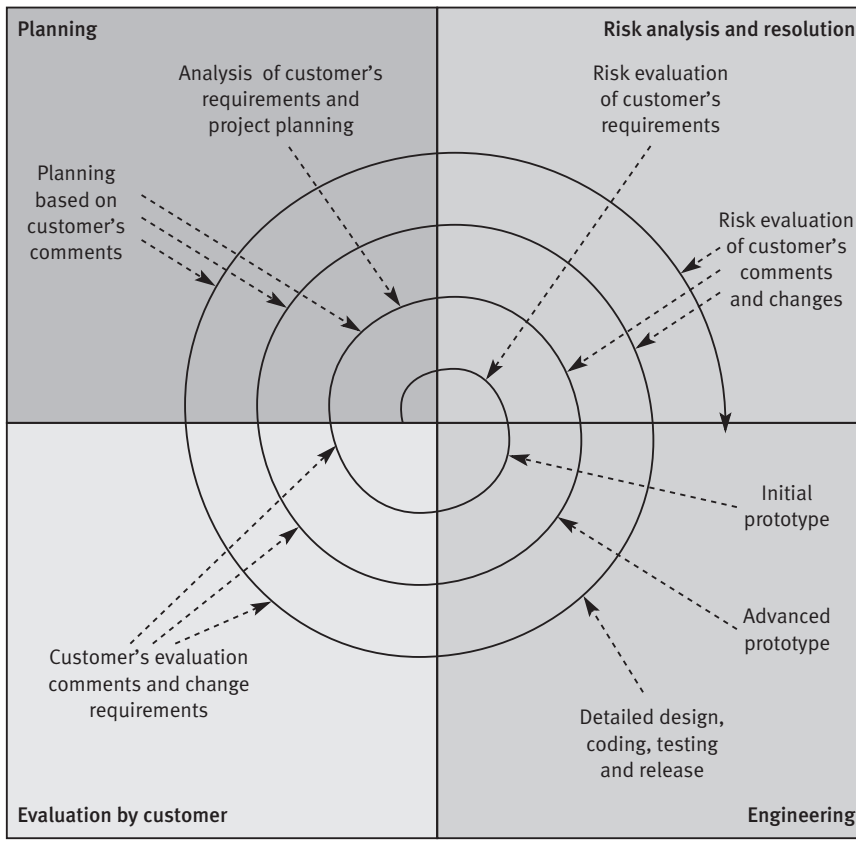


Figure 7.3: The spiral model (Boehm, 1988)
Source: After Boehm (1988) (© 1988 IEEE)

An advanced spiral model, the Win–Win Spiral model (Boehm, 1998), enhances the Spiral model (Boehm, 1988) still further. The advanced model places extra emphasis on communication and negotiation between the customer and the developer. The model's name refers to the fact that by using this process, the customer "wins" in the form of improved chances to receive the system most satisfying to his needs, and the developer "wins" in the form of improved chances to stay within the budget and complete the project by the agreed date. This is achieved by increasing emphasis on customer participation and on engineering activities. These revisions in the development process are shown graphically by two sections of the spiral dedicated to customer participation: the first deals with customer evaluation and the second with customer comments and change requirements. Engineering activity is likewise shown in two sections of the spiral: the first is dedicated to design and the second to construction. By evaluating project progress at the end of each of these sections, the developer is able to better control the entire development process.

Accordingly, in the advanced spiral model, shown in Figure 7.4, the following six activities are carried out in each iteration:

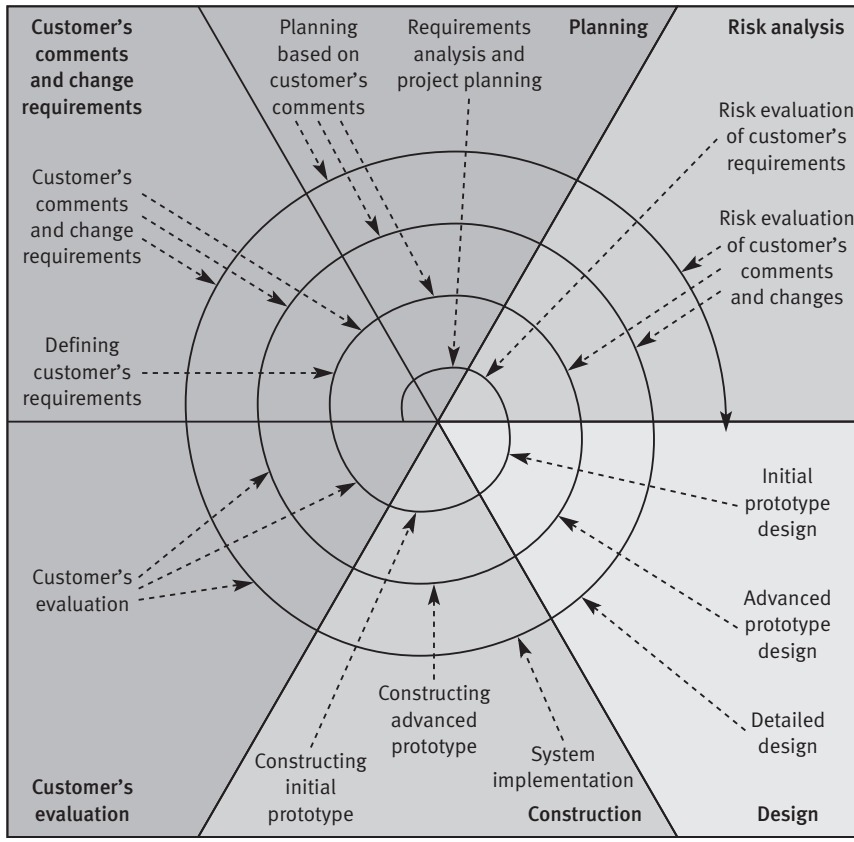


Figure 7.4: The advanced spiral model (Boehm, 1998)

Source: After Boehm (1988) (© 1988 IEEE)

- Customer's specification of requirements, comments and change demands
- Developer's planning activities
- Developer's risk analysis and resolution
- Developer's design activities
- Developer's construction activities pertaining to coding, testing, installation and release
- Customer's evaluation.

7.1.4 The object-oriented model

The object-oriented model differs from the other models by its intensive reuse of software components. This methodology is characterized by its easy integration of existing software modules (called objects or components) into newly developed software systems. A software component library serves this purpose by supplying software components for reuse.

So, according to the object-oriented model as shown in Figure 7.5, the development process begins with a sequence of object-oriented analyses and designs. The design phase is followed by acquisition of suitable components

from the reusable software library, when available. "Regular" development is carried out otherwise. Copies of newly developed software components are then "stocked" in the software library for future reuse. It is expected that the growing software component stocks in the reusable software library will allow substantial and increasing reuse of software, a trend that will allow taking greater advantage of resources as follows:

- Economy – The cost of integrating a reusable software component is much lower than the cost of developing new components.
- Improved quality – Used software components are expected to contain considerably fewer defects than newly developed software components due to detection of faults by former users.

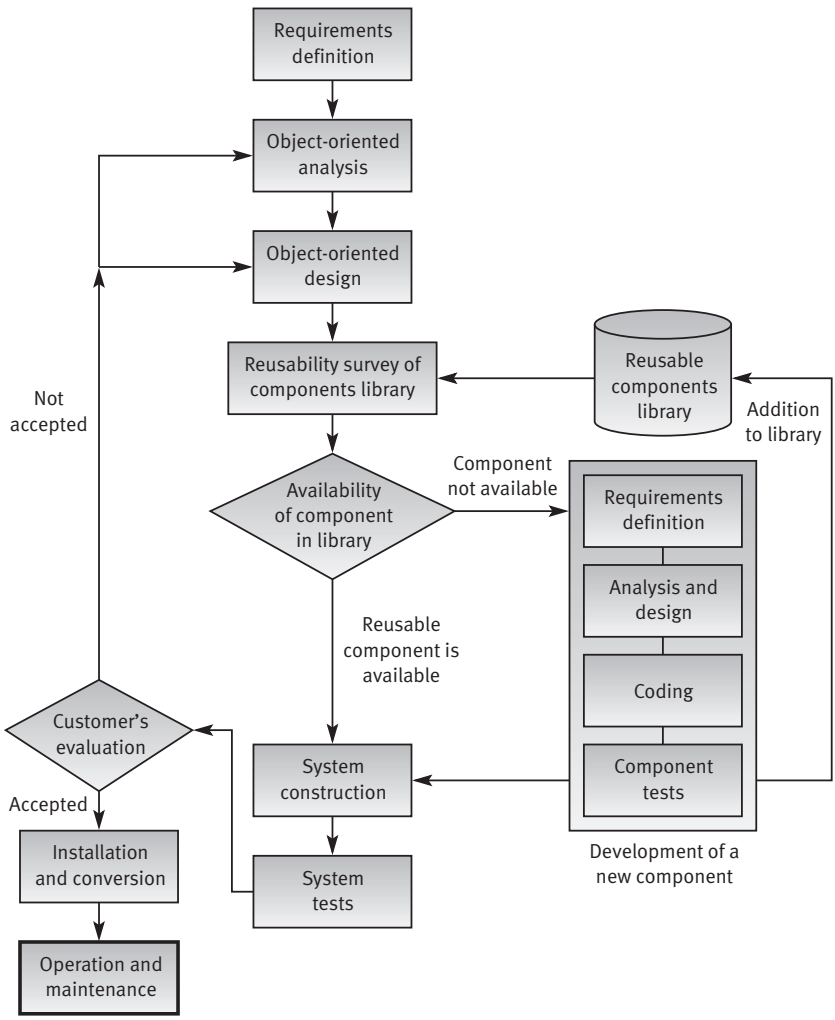


Figure 7.5: The object-oriented model

- Shorter development time – The integration of reusable software components reduces scheduling pressures.

Thus, the advantages of the object-oriented methodology over other methodologies will grow as the storage of reusable software grows.

7.2 Factors affecting intensity of quality assurance activities in the development process

Project life cycle quality assurance activities are process oriented, in other words, linked to completion of a project phase, accomplishment of a project milestone, and so forth. The quality assurance activities will be integrated into the development plan that implements one or more software development models – the waterfall, prototyping, spiral, object-oriented or other models.

Quality assurance planners for a project are required to determine:

- The list of quality assurance activities needed for a project.
- For each quality assurance activity:
  - Timing
  - Type of quality assurance activity to be applied
  - Who performs the activity and the resources required. It should be noted that various bodies may participate in the performance of quality assurance activities: development team and department staff members together with independent bodies such as external quality assurance team members or consultants
  - Resources required for removal of defects and introduction of changes.

Implementation tip

In some development plans, one finds quality assurance activities spread throughout the process, but without any time allocated for their performance or for the subsequent removal of defects. Someone probably assumed that a late afternoon meeting would be sufficient for performing the quality assurance activities and the corrections to be made. As nothing is achieved without time, the almost guaranteed result is delay, caused by the "unexpectedly" long duration of the quality assurance process. Hence, the time allocated for quality assurance activities and the defects correction work that follow should be examined.

The intensity of the quality assurance activities planned, indicated by the number of required activities, is affected by project and team factors, as shown in Frame 7.2.

Frame 7.2 Factors affecting the required intensity of quality assurance activities

Project factors:

- Magnitude of the project
- Technical complexity and difficulty
- Extent of reusable software components
- Severity of failure outcomes if the project fails

Team factors:

- Professional qualification of the team members
- Team acquaintance with the project and its experience in the area
- Availability of staff members who can professionally support the team
- Familiarity with the team members, in other words the percentage of new staff members in the team

The following two examples can illustrate how these factors can influence quality assurance activities.

Example 1

A software development team has planned the quality assurance activities for its new consumer club project. The current project contract, signed with a leading furniture store, is the team's 11th consumer club project dealing in the last three years. The team estimates that about seven man-months need to be invested by the two team members assigned to the project, whose duration is estimated at four months. It is estimated that a reusable components library can supply 90% of the project software.

Three quality assurance activities were planned by the project leader. The quality assurance activities and their duration are listed in Table 7.1.

Table 7.1: Duration of quality assurance activities – the consumer club example

No.	Quality assurance activity	Duration of quality assurance activity (days)	Duration of corrections and changes (days)
1	Design review of requirements definition	0.5	1
2	Inspection of the design	1	1
3	System test of completed software package	4	2

The main considerations affecting this plan are:

- Degree of team acquaintance with the subject
- High percentage of software reuse
- Size of the project (in this case, medium)
- Severity of failure results if the project fails.

Example 2

The real-time software development unit of a hospital's information systems department has been assigned to develop an advanced patient monitoring system. The new monitoring unit is to combine of patient's room unit with a control unit. The patient's room unit is meant to interface with several types of medical equipment, supplied by different manufacturers, which measure various indicators of the patient's condition. A sophisticated control unit will be placed at the nurses' station, with data to be communicated to cellular units carried by doctors.

The project leader estimates that 14 months will be required to complete the system; a team of five will be needed, with an investment of a total of 40 man-months. She estimates that only 15% of the components can be obtained from the reusable component library. The SDLC methodology was chosen to integrate application of two prototypes of the patient's room unit and two prototypes of the control unit for the purpose of improving communication with the users and enhancing feedback of comments at the analysis and design phases.

The main considerations affecting this plan are:

- High complexity and difficulty of the system
- Low percentage of reusable software available
- Large size of the project
- High severity of failure outcomes if the project fails.

The quality assurance activities and their duration, as defined by the project leader, are listed in Table 7.2.

7.3 Verification, validation and qualification

Three aspects of quality assurance of the software product (a report, code, etc.) are examined under the rubrics of verification, validation and qualification.

IEEE Std 610.12-1990 (IEEE, 1990) defines these aspects as follows:

- "**Verification** – The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."

Table 7.2: Duration of quality assurance activities – the patient monitoring system example

No.	Quality assurance activity	Duration of quality assurance activity (days)	Duration of corrections and changes (days)
1	Design review of requirements definition	2	1
2	Design review of analysis of patient's room unit	2	2
3	Design review of analysis of control unit	1	2
4	Design review of preliminary design	1	1
5	Inspection of design of patient's room unit	1	2
6	Inspection of design of control unit	1	3
7	Design review of prototype of patient's room unit	1	1
8	Design review of prototype of control unit	1	1
9	Inspection of detailed design for each software interface component	3	3
10	Design review of test plans for patient's room unit and control unit	3	1
11	Unit tests of software code for each interface module of patient's room unit	4	2
12	Integration test of software code of patient's room unit	3	3
13	Integration test of software code of control unit	2	3
14	System test of completed software system	10	5
15	Design review of user's manual	3	2

- "**Validation** – The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements."
- "**Qualification** – The process used to determine whether a system or component is suitable for operational use."

According to the IEEE definitions, *verification* examines the consistency of the products being developed with products developed in previous phases. When doing so, the examiner follows the development process and assumes that all the former development phases have been completed correctly, whether as originally planned or after removal of all the discovered defects. This assumption forces the examiner to disregard deviations from the customer's original requirements that might have been introduced during the development process.

Validation represents the customer's interest by examining the extent of compliance to his or her original requirements. Comprehensive validation reviews tend to improve customer satisfaction from the system.

Qualification focuses on operational aspects, where maintenance is the main issue. A software component that has been developed and documented

according to professional standards and style and structure convention procedures is expected to be much easier to maintain than one that provides marvelous coding improvisations yet does not follow known coding style procedures and so forth.

Planners are required to determine which of these aspects should be examined in each quality assurance activity.

7.4 A model for SQA defect removal effectiveness and cost

The model deals with two quantitative aspects of an SQA plan consisting of several defect detection activities:

(1) The plan's total effectiveness in removing project defects.
(2) The total costs of removal of project defects.

The plan itself is to be integrated within a project's development process.

7.4.1 The data

The application of the model is based on three types of data, described under the following headings.

Defect origin distribution

Defect origins (the phase in which defects were introduced) are distributed throughout the development process, from the project's initiation to its completion. Surveys conducted by major software developers, such as IBM and TRW, summarized by Boehm (1981, Chapter 24) and Jones (1996, Chapter 3), reveal a similar pattern of defect distribution. Software development professionals believe that this pattern has not changed substantially in the last two decades. A characteristic distribution of software defect origins, based on Boehm (1981) and Jones (1996), is shown in Table 7.3.

Table 7.3: A characteristic distribution of software defect origins

No.	Software development phase	Average percentage of defects originating in phase
1	Requirements specification	15%
2	Design	35%
3	Coding (coding 30%, integration 10%)	40%
4	Documentation	10%

chapter 8

Reviews

Chapter outline

A common product of the software development process, especially in its analysis and design phases, is a design document in which the progress of the development work performed is recorded. The system analyst or analysts who prepared the document will check it repeatedly, it is to be assumed, in order to detect any possible error that might have entered. In addition, development team leaders are also expected to examine this document and its details so as to detect any remaining errors before granting their approval.

However, it is clear that because these professionals were involved in producing the document, they are unlikely to detect some of their own errors irrespective of the number of checks. Therefore, only others – such as peers, superiors, experts, and customer's representatives (those having different experiences and points of view, yet not directly involved in creating the document) – are capable of reviewing the product and detecting the errors unnoticed by the development team.

As defined by IEEE (1990), a review process is:

> "A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval."

As these documents are products of the project's initial phases, reviews acquire special importance in the SQA process because they provide early detection and prevent the passing of design and analysis errors "downstream", to stages where error detection and correction are much more intricate, cumbersome, and therefore costly.

Several methodologies can be implemented when reviewing documents. In this chapter, the following methods will be discussed:

- Formal design reviews
- Peer reviews (inspections and walkthroughs)
- Expert opinions.

Standards for software reviews are the subject of IEEE Std 1028 (IEEE, 1997).

It should be noted that successful implementations of inspections and walkthroughs also detect defects in the coding phase, where the appropriate document reviewed is the code printout.

A case study of the contribution of formal design reviews and inspections to software quality is presented by MacFarland (2001).

After completing this chapter, you will be able to:

- Explain the direct and indirect objectives of review methodologies.
- Explain the contribution of external experts to the performance of review tasks.
- Compare the three major review methodologies.

8.1 Review objectives

Several objectives *motivate* reviews. The review's *direct objectives* deal with the current project, whereas its *indirect objectives*, more general in nature, deal with the contribution *of the review proper* to the promotion of team members' professional knowledge and the improvement of the development methodologies applied by the organization.

The main review objectives are presented in Frame 8.1.

Frame 8.1 Review objectives

Direct objectives

- To detect analysis and design errors as well as subjects where corrections, changes and completions are required with respect to the original specifications and approved changes.
- To identify new risks likely to affect completion of the project.
- To locate deviations from templates and style procedures and conventions. Correction of these deviations is expected to contribute to improved communication and coordination resulting from greater uniformity of methods and documentation style.
- To approve the analysis or design product. Approval allows the team to continue to the next development phase.

Indirect objectives

- To provide an informal meeting place for exchange of professional knowledge about development methods, tools and techniques.
- To record analysis and design errors that will serve as a basis for future corrective actions. The corrective actions are expected to improve development methods by increasing effectiveness and quality, among other product features. (For more about corrective actions, see Chapter 17.)

The various review methods differ in the emphasis attached to the different objectives and in the extent of success achievable for each objective. Therefore, for better "filtering out" of errors and greater long-term impacts, a double or even triple "net", constructed from among the range of review methods available, should be applied.

Reviews are not activities to be conducted haphazardly. Procedural order and teamwork lie at the heart of formal design reviews, inspections and walkthroughs. Each participant is expected to emphasize his or her area of responsibility or specialization when making comments. At each review session, one individual is assigned the task of inscribing mutually agreed remarks. The subsequent list of items should include full details of defect location and description, documented in a way that will later allow full retrieval by the development team. However, because of the human propensity to try to design solutions on the spot and, often, to digress to tangential issues or, even worse, to personal matters during the course of a meeting, a coordinator is needed to maintain control of the discussion and keep it on track.

In general, the knowledge that an analysis or design product will be reviewed stimulates the development team to work at their maximum. This represents a further contribution of reviews to improved product quality.

In the following, the various review methods are presented. A comparison of team review methods is the subject of Section 8.4; expert opinions are discussed in Section 8.5.

8.2 Formal design reviews (DRs)

Formal design reviews, variously called "design reviews", "DRs" and "formal technical reviews (FTR)", differ from all other review instruments by being the only reviews that are necessary for approval of the design product. Without this approval, the development team cannot continue to the next phase of the software development project. Formal design reviews may be conducted at any development milestone requiring completion of an analysis or design document, whether that document is a requirement specification or an installation plan. A list of common formal design reviews is given in Frame 8.2.

Frame 8.2 Some common formal design reviews

DPR – Development Plan Review
SRSR – Software Requirement Specification Review
PDR – Preliminary Design Review
DDR – Detailed Design Review
DBDR – Data Base Design Review
TPR – Test Plan Review
STPR – Software Test Procedure Review
VDR – Version Description Review
OMR – Operator Manual Review
SMR – Support Manual Review
TRR – Test Readiness Review
PRR – Product Release Review
IPR – Installation Plan Review

Sauer and Jeffery (2000) discuss a broad range of factors affecting the effectiveness of DRs, based on research results and a wide-ranging survey of the literature. Our discussion of formal design reviews will focus on:

- The participants
- The prior preparations
- The DR session
- The recommended post-DR activities.

8.2.1 The participants in a DR

All DRs are conducted by a review leader and a review team. The choice of appropriate participants is of special importance because of their power to approve or disapprove a design product.

The review leader

Because the appointment of an appropriate review leader is a major factor affecting the DR's success, certain characteristics are to be looked for in a candidate for this position:

- Knowledge and experience in development of projects of the type reviewed. Preliminary acquaintance with the current project is not necessary.
- Seniority at a level similar to if not higher than that of the project leader.
- A good relationship with the project leader and his team.
- A position external to the project team.

Thus, appropriate candidates for review team leadership include the development department's manager, the chief software engineer, the leader of another project, the head of the software quality assurance unit and, in certain circumstances, the customer's chief software engineer.

Implementation tip

In some cases, the project leader is appointed as the review leader, the main justification for this decision being his or her superior knowledge of the project's material. In most cases, this choice proves to be undesirable professionally. A project leader who serves as the review team leader tends, whether intentionally or nor, to limit the scope of the review and avoid incisive criticism. Review team members tend to be chosen accordingly. Appointments of this type usually undermine the purpose for the review and only delay confrontation with problems to a later, more sensitive date.

Small development departments and small software houses typically have substantial difficulties finding an appropriate candidate to lead the review team. One possible solution to this predicament is the appointment of an external consultant to the position.

The review team

The entire review team should be selected from among the senior members of the project team together with appropriate senior professionals assigned to other projects and departments, customer–user representatives, and in some cases, software development consultants. It is desirable for non-project staff to make up the majority of the review team.

An important, oft-neglected issue is the size of the review team. A review team of three to five members is expected to be an efficient team, given the proper diversity of experience and approaches among the participants are assured. An excessively large team tends to create coordination problems, waste review session time and decrease the level of preparation, based on a natural tendency to assume that others have read the design document.

8.2.2 Preparations for a DR

Although preparations for a DR session are to be completed by all three main participants in the review – the review leader, the review team and the development team – each participant is required to focus on distinct aspects of the process.

Review leader preparations

The main tasks of the review leader in the preparation stage are:

- To appoint the team members
- To schedule the review sessions
- To distribute the design document among the team members (hard copy, electronic file, etc.).

It is of utmost importance that the review session be scheduled shortly after the design document has been distributed to the review team members. Timely sessions prevent an unreasonable length of time from elapsing before the project team can commence the next development phase and thus reduce the risk of going off schedule.

Review team preparations

Team members are expected to review the design document and list their comments prior to the review session. In cases where the documents are sizable, the review leader may ease the load by assigning to each team member review of only part of the document.

An important tool for ensuring the review's completeness is the checklist. In addition to the general design review checklist, checklists dedicated to the more common analysis and design documents are available and can be constructed when necessary. Checklists contribute to the design review's effectiveness by reminding the reviewer of all the primary and secondary issues requiring attention. For a comprehensive discussion of checklists, see Chapter 15.

Development team preparations

The team's main obligation as the review session approaches is to prepare a short presentation of the design document. Assuming that the review team members have read the design document thoroughly and are now familiar

with the project's outlines, the presentation should focus on the main professional issues awaiting approval rather than wasting time on description of the project in general.

Implementation tip

One of the most common techniques used by project leaders to avoid professional criticism and undermine review effectiveness is the comprehensive presentation of the design document. This type of presentation excels in the time it consumes. It exhausts the review team and leaves little time, if any, for discussion. All experienced review leaders know how to handle this phenomenon.

In cases where the project leader serves as the review leader, one can observe especially potent tactics aimed at stymieing an effective review: appointment of a large review team combined with a comprehensive and long presentation.

8.2.3 The DR session

The review leader's experience in leading the discussions and sticking to the agenda is the key to a successful DR session. A typical DR session agenda includes:

(1) A short presentation of the design document.
(2) Comments made by members of the review team.
(3) Verification and validation in which each of the comments is discussed to determine the required actions (corrections, changes and additions) that the project team has to perform.
(4) Decisions about the design product (document), which determines the project's progress. These decisions can take three forms:
   - *Full approval* – enables immediate continuation to the next phase of the project. On occasion, full approval may be accompanied by demands for some minor corrections to be performed by the project team.
   - *Partial approval* – approval of immediate continuation to the next phase for some parts of the project, with major action items (corrections, changes and additions) demanded for the remainder of the project. Continuation to the next phase of these remainder parts will be permitted only after satisfactory completion of the action items. This approval can be given by the member of the review team assigned to review the completed action items, by the full review team in a special review session, or by any other forum the review leader thinks appropriate.
   - *Denial of approval* – demands a repeat of the DR. This decision is applied in cases of multiple major defects, particularly critical defects.

8.2.4 Post-review activities

Apart from the DR report, the DR team or its representative is required to follow up performance of the corrections and to examine the corrected sections.

The DR report

One of the review leader's responsibilities is to issue the DR report immediately after the review session. Early distribution of the DR report enables the development team to perform the corrections earlier and minimize the attendant delays to the project schedule.

The report's major sections contain:

- A summary of the review discussions.
- The decision about continuation of the project.
- A full list of the required actions – corrections, changes and additions that the project team has to perform. For each action item, the anticipated completion date and project team member responsible are listed.
- The name(s) of the review team member(s) assigned to follow up performance of corrections.

The form shown in Appendix 8A presents the data items that need to be documented for an inclusive DR report.

The follow-up process

The person appointed to follow up the corrections, in many cases the review leader him or herself, is required to determine whether each action item has been satisfactorily accomplished as a condition for allowing the project to continue to the next phase. Follow-up should be fully documented to enable clarification of the corrections in the future, if necessary.

Implementation tip

Unfortunately, the entire or even parts of the DR report are often worthless, whether because of an inadequately prepared review team or because of intentional evasion of a thorough review. It is fairly easy to identify such cases from the characteristics of the review report:

- An extremely short report, limited to documented approval of the design product, listing no detected defects.
- A short report, approving continuation to the next project phase in full, listing several minor defects but no action items.
- A report listing several action items of varied severity, but no indication of follow-up (correction schedule, etc.), and no available documented follow-up activities.

Pressman (2000, Chapter 8) lists guidelines for completing a successful DR, while focusing on infrastructure, preparations for a DR, and conduct of a DR session are summarized in Frame 8.3. Pressman's golden "guidelines" for formal design reviews also apply to inspection and walkthrough sessions.

Frame 8.3 **Pressman's 13 "golden guidelines" for a successful design review (based on Pressman 2000, Chapter 8)**

Design review infrastructure

- Develop checklists for each type of design document, or at least for the common ones.
- Train senior professionals to treat major technical as well as review process issues. The trained professionals serve as a reservoir for DR teams.
- Periodically analyze past DR effectiveness regarding defect detection to improve the DR methodology.
- Schedule the DRs as part of the project activity plan and allocate the needed resources as an integral part of the software development organization's standard operating procedures.

The design review team

- Review teams should be limited in size, with 3–5 members usually being the optimum.

The design review session

- Discuss professional issues in a constructive way while refraining from personalizing those issues. This demands keeping the discussion atmosphere free of unnecessary tension.
- Keep to the review agenda. Drifting from the planned agenda usually interferes with the review's efficiency.
- Focus on detection of defects by verifying and validating the participants' comments. Refrain from discussing possible solutions to the detected defects so as to save time and avoid wandering from the agenda.
- In cases of disagreement about the significance of an error, it is desirable to end the debate by noting the issue and shifting its discussion to another forum.
- Properly document the discussions, especially details of the participants' comments and the results of their verification and validation. This step is especially important if the documentation is to serve as input or a basis for preparation of the review report.
- The duration of a review session should not exceed two hours.

Post-review activities

- Prepare the review report, which summarizes the issues discussed and the action items.
- Establish follow-up procedures to ensure the satisfactory performance of all the corrections included in the list of action items.

The formal design review process is illustrated in Figure 8.1.

The next section deals with peer review methods, and discusses the two most commonly used methods: inspection and walkthrough.

8.3 Peer reviews

Two peer review methods, inspections and walkthroughs, are discussed in this section. The major difference between formal design reviews and peer review methods is rooted in their participants and authority. While most participants in DRs hold superior positions to the project leader and customer representatives, participants in peer reviews are, as expected, the project leader's equals, members of his or her department and other units. The other major difference lies in degree of authority and the objective of each review method. Formal design reviews are authorized to approve the design document so that work on the next stage of the project can begin. This authority is not granted to the peer reviews, whose main objectives lie in detecting errors and deviations from standards.

Today, with the appearance of computerized design tools, including CASE tools, on the one hand, and systems of vast software packages on the other hand, some professionals tend to diminish the value of manual reviews such as inspections and walkthroughs. Nevertheless, past software surveys as well as recent empirical research findings provide much convincing evidence that peer reviews are highly efficient as well as effective methods.

What differentiates a walkthrough from an inspection is the level of formality, with inspection the more formal of the two. Inspection emphasizes the objective of corrective action. Whereas a walkthrough's findings are limited to comments on the document reviewed, an inspection's findings are also incorporated into efforts to improve development methods *per se*. Inspections, as opposed to walkthroughs, are therefore considered to contribute more significantly to the general level of SQA.

Inspection is usually based on a comprehensive infrastructure, including:

- Development of inspection checklists developed for each type of design document as well as coding language and tool, which are periodically updated.
- Development of typical defect type frequency tables, based on past findings, to direct inspectors to potential "defect concentration areas".

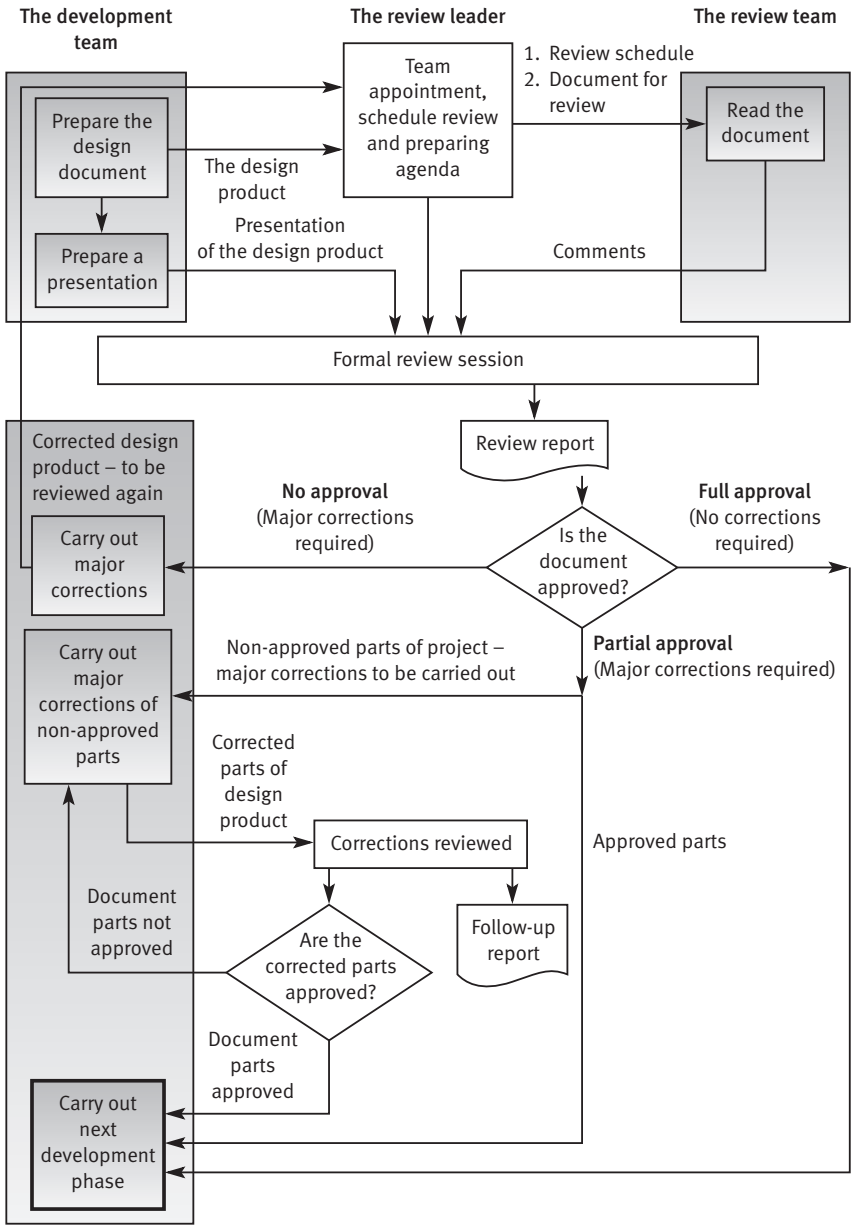


Figure 8.1: The formal design review process

- Training of competent professionals in inspection process issues, a process that makes it possible for them to serve as inspection leaders (moderators) or inspection team members. The trained employees serve as a reservoir of professional inspectors available for future projects.

- Periodic analysis of the effectiveness of past inspections to improve the inspection methodology.
- Introduction of scheduled inspections into the project activity plan and allocation of the required resources, including resources for correction of detected defects.

The inspection and walkthrough processes described here are the more commonly employed versions of these methods. Organizations often modify these methods, with adaptations representing "local color", that is, the character of the development and SQA units, the software products developed, team structure and composition, and the like. It should be noted that in response to this variability, especially in walkthrough procedures, differences between the two methods are easily blurred. This state of affairs has convinced some specialists to view walkthroughs as a type of inspection, and vice versa.

The debate over which method is preferable has yet to be resolved, with proponents of each arguing for the superiority of their favored approach. Based on their survey of studies of each method, Gilb and Graham (1993) conclude that as an alternative to inspections, walkthroughs display "far fewer defects found but at the same cost".

Our discussion of peer review methods will thus focus on:

- Participants of peer reviews
- Requisite preparations for peer reviews
- The peer review session
- Post-peer review activities
- Peer review efficiency
- Peer review coverage.

With minor adaptations, the principles and process of design peer reviews can also be successfully applied to code peer reviews.

Design and code inspections, as procedural models, were initially described and formulated by Fagan (1976, 1986). As to walkthroughs, Yourdon (1979) provides a thorough and detailed discussion of the related principles and processes.

8.3.1 Participants of peer reviews

The optimal peer review team is composed of three to five participants. In certain cases, the addition of one to three further participants is acceptable. All the participants should be peers of the software system designer-author. A major factor contributing to the success of a peer review is the group's "blend" (which differs between inspections and walkthroughs).

A recommended peer review team includes:

- A review leader
- The author
- Specialized professionals.

The review leader

The role of review leader ("moderator" in inspections, "coordinator' in walkthroughs) differs only slightly by peer review type. Candidates for this position must:

(1) Be well versed in development of projects of the current type and familiar with its technologies. Preliminary acquaintance with the current project is not necessary.
(2) Maintain good relationships with the author and the development team.
(3) Come from outside the project team.
(4) Display proven experience in coordination and leadership of professional meetings.
(5) For inspections, training as a moderator is also required.

The author

The author is, invariably a participant in each type of peer review.

Specialized professionals

The specialized professionals participating in the two peer review methods differ by review. For inspections, the recommended professionals are:

- **A designer:** the systems analyst responsible for analysis and design of the software system reviewed.
- **A coder or implementer:** a professional who is thoroughly acquainted with coding tasks, preferably the leader of the designated coding team. This inspector is expected to contribute his or her expertise to the detection of defects that could lead to coding errors and subsequent software implementation difficulties.
- **A tester:** an experienced professional, preferably the leader of the assigned testing team, who focuses on identification of design errors usually detected during the testing phase.

For walkthroughs, the recommended professionals are:

- **A standards enforcer.** This team member, who specializes in development standards and procedures, is assigned the task of locating deviations from those standards and procedures. Errors of this type substantially affect the team's long-term effectiveness, first because they cause extra difficulties for new members joining the development team, and later because they will reduce the effectiveness of the team that will maintain the system.
- **A maintenance expert** who is called upon to focus on maintainability, flexibility and testability issues (see Chapter 3), and to detect design defects capable of impeding correction of bugs or performance of future changes. Another area requiring his or her expertise is documentation, whose completeness and correctness are vital for any maintenance activity.

- **A user representative.** Participation of an internal (when the customer is a unit in the same firm) or an external user's representative in the walkthrough team contributes to the review's validity because he or she examines the software system from the point of view of the user-consumer rather than the designer–supplier. In cases where a "real" user is not available, as in the development of a COTS software package, a team member may take on that role and focus on validity issues by comparing of the original requirements with the actual design.

Team assignments

Conducting a review session requires, naturally, assignment of specific tasks to the team members. Two of these members are the presenter of the document and the scribe, who documents the discussions.

- **The presenter.** During inspection sessions, the presenter of the document is chosen by the moderator; usually, the presenter is not the document's author. In many cases the software coder serves as the presenter because he or she is the team member who is most likely to best understand the design logic and its implications for coding. In contrast, for most walkthrough sessions, it is the author, the professional most intimately acquainted with the document, who is chosen to present it to the group. Some experts claim that an author's assignment as presenter may affect the group members' judgement; therefore, they argue that the choice of a "neutral" presenter is to be preferred.
- **The scribe.** The team leader will often – but not always – serve as the scribe for the session, and record the noted defects that are to be corrected by the development team. This task is more than procedural; it requires thorough professional understanding of the issues discussed.

8.3.2 Preparations for a peer review session

The review leader and the team members are to assiduously complete their preparation, with the type of review determining their scope.

Peer review leader's preparations for the review session

The main tasks of the review leader in the preparation stage are:

- To determine, together with the author, which sections of the design document are to be reviewed. Such sections can be:
  - The most difficult and complex sections
  - The most critical sections, where any defect can cause severe damage to the program application and thus to the user
  - The sections prone to defects.
- To select the team members.

- To schedule the peer review sessions. It is advisable to limit a review session to two hours; therefore, several review sessions should be scheduled (up to two sessions a day) when the review task is sizable. It is important to schedule the sessions shortly after the pertinent design document sections are ready for inspection. This proximity tends to minimize the scope and/or number of design additions based on parts of the document that might be found defective later in the scheduled review. Moreover, for the process to unfold smoothly, the inspection's review leader should schedule an overview meeting for his team.
- To distribute the document to the team members prior to the review session.

Peer review team's preparations for the review session

The preparations required of an inspection team member are quite thorough, while those required of a walkthrough team member are brief.

Inspection team members are expected to read the document sections to be reviewed and list their comments before the inspection session begins. This advance preparation is meant to guarantee the session's effectiveness. They will also be asked to participate in an overview meeting. At this meeting, the author provides the inspection team members with the necessary relevant background for reviewing the chosen document sections: the project in general, the logic, processes, outputs, inputs, and interfaces. In cases where the participants are already well acquainted with the material, an overview meeting may be waived.

An important tool supporting the inspector's review is a checklist. In well-established development departments, one can find specialized checklists dedicated to the more common types of development documents (see Chapter 15).

Prior to the walkthrough session, team members briefly read the material in order to obtain a general overview of the sections to be reviewed, the project and its environment. Participants lacking preliminary knowledge of the project and its substantive area will need far more preparation time. In most organizations employing walkthroughs, team participants are not required to prepare their comments in advance.

8.3.3 The peer review session

A typical peer review session takes the following form. The presenter reads a section of the document and adds, if needed, a brief explanation of the issues involved in his or her own words. As the session progresses, the participants either deliver their comments to the document or address their reactions to the comments. The discussion should be confined to identification of errors, which means that it should not deal with tentative solutions. Unlike inspection sessions, the agenda of the typical walkthrough session opens with the author's short presentation or overview of the project and the design sections to be reviewed.

During the session, the scribe should document each error recognized by location and description, type and character (incorrect, missing parts or extra parts). The inspection session scribe will add the estimated severity of each defect, a factor to be used in the statistical analysis of defects found and for the formulation of preventive and corrective actions. The error severity classification appearing in Appendix C of MIL-STD-498 (DOD, 1994) and presented in Table 8.1, provides an accepted framework for classifying error severity.

Concerning the length of inspection and walkthrough sessions, the same rules apply as to DRs: sessions should not exceed two hours in length, or schedule for more than twice daily. Pressman's "golden guidelines" for conducting successful DR sessions are also helpful here (see Frame 8.3).

Session documentation

The documentation produced at the end of an inspection session is much more comprehensive than that of a walkthrough session.

Two documents are to be produced following an inspection session and subsequently distributed among the session participants:

(1) **Inspection session findings report.** This report, produced by the scribe, should be completed and distributed immediately after the session's closing. Its main purpose is to assure full documentation of identified errors for correction and follow up. An example of such a report is provided in Appendix 8B.

Table 8.1: Classification of design errors by severity

Severity	Description
5 (critical)	(1) Prevents accomplishment of essential capabilities. (2) Jeopardizes safety, security or other critical requirements.
4	(1) Adversely affects the accomplishment of essential capabilities, where no work-around solution is known. (2) Adversely affects technical, cost or schedule risks to project or system maintenance, where no work-around solution is known.
3	(1) Adversely affects the accomplishment of essential capabilities, where a work-around solution is known. (2) Adversely affects technical, cost or schedule risks to the development project or to the system maintenance, where a work-around solution is known.
2	(1) User/operator inconvenience that does not affect required mission or operational essential capabilities. (2) Inconvenience for development or maintenance personnel, but does not prevent the realization of those responsibilities.
1 (minor)	Any other effect.

Source: After DOD (1994)

(2) **Inspection session summary report.** This report is to be compiled by the inspection leader shortly after the session or series of sessions dealing with the same document. A typical report of this type summarizes the inspection findings and the resources invested in the inspection; it likewise presents basic quality and efficiency metrics. The report serves mainly as input for analysis aimed at inspection process improvement and corrective actions that go beyond the specific document or project. An example of an inspection session summary report appears in Appendix 8C.

At the end of a session or series of walkthrough sessions, copies of the error documentation – the "walkthrough session findings report" – should be handed to the development team and the session participants.

8.3.4 Post-peer review activities

A fundamental element differentiating between the two peer review methods discussed here is the issue of post-peer review.

The inspection process, contrary to the walkthrough, does not end with a review session or the distribution of reports. Post-inspection activities are conducted to attest to:

- The prompt, effective correction and reworking of all errors by the designer/author and his team, as performed by the inspection leader (or other team member) in the course of the assigned follow-up activities.
- Transmission of the inspection reports to the internal Corrective Action Board (CAB) for analysis. This action initiates the corrective and preventive actions that will reduce future defects and improve productivity (see Chapter 17).

A comparison of the peer review methods, participants and process elements is presented in Figure 8.2.

8.3.5 The efficiency of peer reviews

The issue of defect detection efficiency of peer review methods proper and in comparison to other SQA defect detection methods is constantly being debated. Some of the more common metrics applied to estimate the efficiency of peer reviews, as suggested in the literature, are:

- Peer review detection efficiency (average hours worked per defect detected).
- Peer review defect detection density (average number of defects detected per page of the design document).
- Internal peer review effectiveness (percentage of defects detected by peer review as a percentage of total defects detected by the developer).

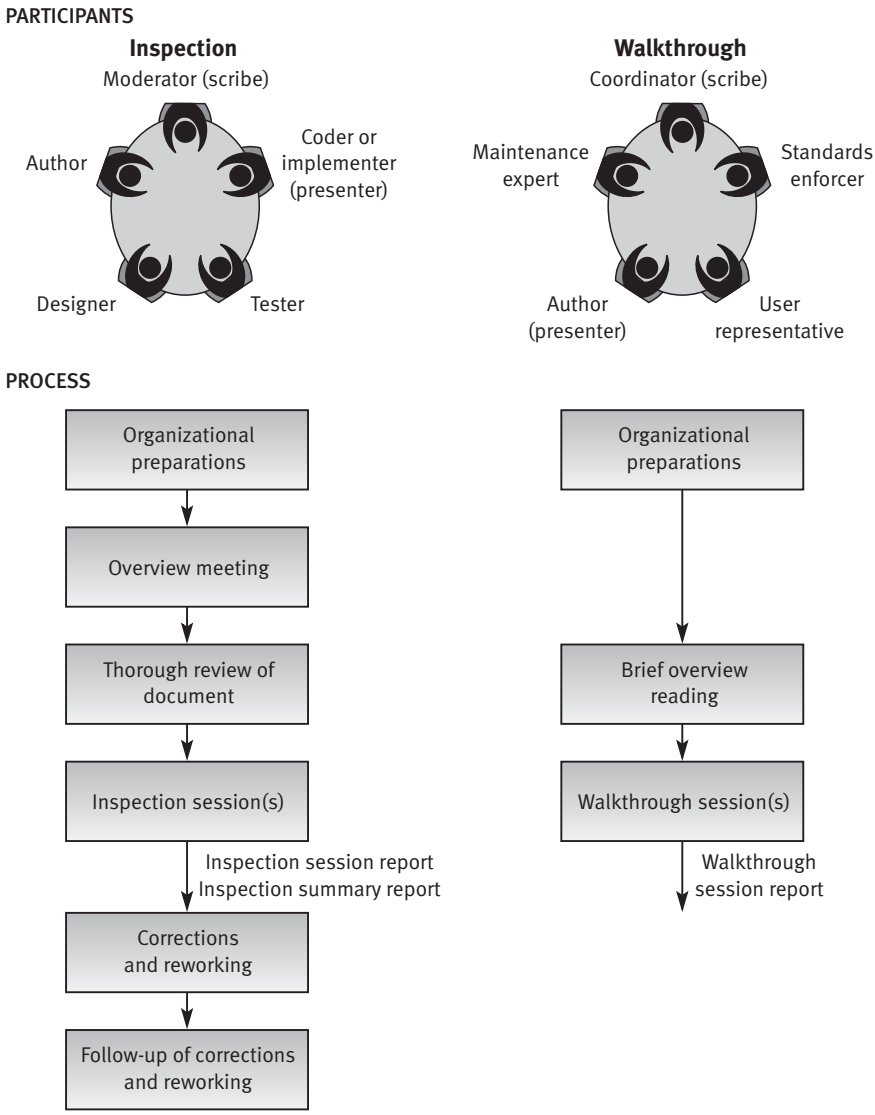


Figure 8.2: Inspection vs. walkthrough – participants and processes

The literature provides rather meager indications about findings inspection effectiveness. Dobbins (1998) quotes Madachy's findings from an analysis of the design and code inspections conducted on the Litton project. Madachy's findings regarding the first two metrics cited above are presented in Table 8.2.

Dobbins (1998) also cites Don O'Neill's 1992 National Software Quality Experiment, conducted in 27 inspection laboratories operating in the US. This experiment provides some insight into the code inspection process, especially at the preparation stage. A total of 90 925 source code lines were code-inspected, with the following results:

Table 8.2: The Litton project's inspection efficiency according to Madachy

Type of document	No. of inspections	Total number of defects and major defects	No. of pages	Inspection resouces invested (work hours)	Inspection efficiency metrics: Defect detection density (defects/page)	Inspection efficiency metrics: Inspection detection efficiency (work-hours/major defect)
Design inspections						
Requirements description	21	1243 (89 major)	552	328	2.25	3.69
Requirements analysis	32	2165 117 major	1065	769	2.03	6.57
High-level design	41	2398 (197 major)	1652	1097	1.45	5.57
Test procedures	18	1495 (121 major)	1621	457	0.92	3.78
Code inspections						
Code	150	7165 (772 major)	5047*	4612	1.42	5.97

*276 422 lines of code.
Source: After Dobbins (1998)

- Total number of defects detected 1849
- Number of major defects detected 242
- Total preparation time (minutes) 22 828

Accordingly:

- Average preparation time per detected defect
  12.3 minutes (0.2 hours)
- Average preparation time per detected major defect
  94.3 minutes (1.57 hours)

Considering the different environments, a comparison of the defect densities detected in the National Software Quality Experiment and those found in the Litton project reveal relatively small differences, as shown below:

	National Software Quality Experiment	Litton Project
Total defect detection density (defects per KLOC*)	20.3	25.9
Major defect detection density (defects per KLOC*)	2.66	2.80

*KLOC = 1000 lines of code.

The internal effectiveness of inspections is discussed by Cusumano (1991, pp. 352–353), who reports the results of a study on the effectiveness of design review, code inspection and testing at Fujitsu (Japan) for the period 1977–1982. After two decades, the findings are still of interest, even though no efficiency metrics are provided. A comparison by year of inspection, presented in Table 8.3, shows substantial improvement in software quality associated with an increased share of code inspection and design reviews and a reduced share of software testing. The software quality is measured here by the number of defects per 1000 lines of maintained code, detected by the users during the first six months of regular software system use.

Though quantitative research results refer only to the inspection method, we can expect to obtain similar results after application of the walkthrough method. This assumption will one day have to be verified empirically for us to be certain.

8.3.6 Peer review coverage

Only a small percentage of the documents and total volume of code ever undergoes peer review. Coverage of about 5–15% of document pages still represents a significant contribution to total design quality because the factor that determines the benefits of peer review to total quality is not the percentage of pages covered but the choice of those pages. Importantly, with the increased usage of reused software, the number of document pages and code lines demanding inspection is obviously declining. Frame 8.4 lists those document sections that are recommended for inclusion in a peer review as well as those that can be readily omitted.

8.4 A comparison of the team review methods

For practitioners and analysts alike, a comparison of the three team review methods discussed in this chapter should prove interesting. Table 8.4 presents such a comparison.

Table 8.3: Code inspection effectiveness at Fujitsu according to Cusumano

Year	Defect detection method			Defects per 1000
	Test %	Design review %	Code inspection %	lines of maintained code
1977	85	–	15	0.19
1978	80	5	15	0.13
1979	70	10	20	0.06
1980	60	15	25	0.05
1981	40	30	30	0.04
1982	30	40	30	0.02

Source: After Cusumano (1991)

chapter 11

Assuring the quality of software maintenance components

Chapter outline

The major part of the software life cycle is the operation period, usually lasting for 5 to 10 years, although cases of software being operational for 15 years and even more are not rare. What makes one software package capable of reaching "old age" with satisfied users, while another package, serving almost the same population, "perishes young"? The main factor responsible for long and successful service is the quality of maintenance. Just how important software maintenance is can be surmised by the attention given the subject in the ISO 9000-3 Standard (see ISO (1997), Sec. 4.19 and ISO/IEC (2001), Sec. 7.5), IEEE (1998) and Oskarsson and Glass (1996).

This chapter will therefore pursue the following quality assurance issues as they relate to software maintenance:

- The foundations for high quality maintenance
- Pre-maintenance software quality components
- SQA tools for corrective maintenance
- SQA tools for functionality improvement maintenance
- Infrastructure SQA tools for software maintenance
- Managerial control SQA tools for software maintenance.

After completing this chapter, you will be able to:

- List software maintenance components and explain their distinction.
- Explain the foundations of high quality maintenance.
- Describe and explain pre-maintenance software quality components.
- List the infrastructure tools that support maintenance quality assurance.
- List the managerial tools for controlling software maintenance quality and explain their importance.

11.1 Introduction

The following three components of maintenance service are all essential for success:

- **Corrective maintenance** – user support services and software corrections.
- **Adaptive maintenance** – adapts the software package to differences in new customer requirements, changing environmental conditions and the like.
- **Functionality improvement maintenance** – combines (1) **perfective maintenance** of new functions added to the software so as to enhance performance, with (2) **preventive maintenance** activities that improve reliability and system infrastructure for easier and more efficient future maintainability.

The inclusion of user support services ("user support centers") in corrective maintenance may need some clarification. User support services is the address for solution of all user difficulties arising when using the software system; software correction services are usually integrated in this service. The user's difficulties may have been caused by:

- Code failure (usually termed "software failure").
- Documentation failure in the user's manual, help screens or other form of documentation prepared for the user. In this case, the support service can provide the user with correct instructions (although no correction of the software documentation itself is performed).
- Incomplete, vague or imprecise documentation.
- User's insufficient knowledge of the software system or his or her failure to use the documentation supplied. In these situations no software system failure is encountered.

The first three of the above causes are considered software system failures. In addition, integration of user support services and software correction services is generally accomplished in close cooperation, with much sharing of information. The other components of maintenance services – functionality improvement and adaptive maintenance – tend not to be initiated by the user support services. In most cases, the functionality improvement and adaptive tasks display the characteristics of a small or large project, depending on the customer's needs. This being the case, these tasks can be performed by a software development unit as well. Considering the above, it is reasonable to include user support services among the corrective maintenance activities.

Generally, one may say that while corrective maintenance ensures that current users can operate the system as specified, adaptive maintenance enables expansion of the user population, while functionality improvement maintenance extends the package's service period.

As mentioned in previous chapters, the combination of the three components of software maintenance consumes more than 60% of total design and programming resources invested in a software system throughout its life cycle (Pressman, 2000). Others estimate that the share of maintenance resources ranges from over 50% (Lientz and Swanson, 1980) to about 65–75% (McKee, 1984) of total project development resources.

The distribution of maintenance resources to the various maintenance services is estimated as follows:

Maintenance service	Lientz and Swanson (1980)	Oskarsson and Glass (1996)
Corrective maintenance	22%	17%
Adaptive maintenance	24%	23%
Functionality improvement maintenance	54%	60%

Surveys of this issue are rare; however, the figures reported by Nosek and Palvia (1990) do not significantly diverge from the estimates shown here. It is believed that the 1980 figures, with minimal changes, continue to represent actual distribution.

The objectives of software maintenance QA activities are presented in Frame 11.1 (repeated from Frame 2.7).

As the nature of the different types of software maintenance components varies substantially, so do the required quality assurance tools. In general, functionality improvement maintenance activities, most adaptive maintenance activities and the software development process basically share the same software quality assurance tools. However, SQA tools employed for corrective maintenance tend to display some unique characteristics. It is important to remember that corrective maintenance activities are service activities and that, unlike functionality improvement and adaptive tasks, they are performed under the close supervision of the user/customer. Management of corrective maintenance services focuses mainly on the availability of services

Frame 11.1 Software maintenance QA activities: objectives

1. Assure, with an accepted level of confidence, that the software maintenance activities conform to the functional technical requirements.
2. Assure, with an accepted level of confidence, that the software maintenance activities conform to managerial scheduling and budgetary requirements.
3. Initiate and manage activities to improve and increase the efficiency of software maintenance and SQA activities. This involves improving the prospects of achieving functional and managerial requirements while reducing costs.

and their quality (measured by time to solution, percentage of cases of correction failures, etc.) rather than on the budgetary and timetable controls typically applied when managing functionality improvement and adaptive maintenance tasks.

General discussion of a variety of software maintenance issues took place at the IEEE International Conference in Oxford, England (IEEE Computer Society, 1999).

11.2 The foundations of high quality

It goes without saying that the quality of the software package to be maintained is perhaps the single most important foundation underlying the quality of maintenance services. Another critical foundation is maintenance policy. The discussion of these subjects follows.

11.2.1 Foundation 1: software package quality

The quality of the software package that is to be maintained clearly stems from the expertise and efforts of the development team as well as the SQA activities performed throughout the development process. If the quality of the package is poor, maintenance will be poor or ineffective, almost by definition. In light of this fundamental insight, we choose to stress here those seven of the original 11 quality assurance factors (see Chapter 3) that have a direct impact on software maintenance. Specifically, we will be discussing two of the five product operation factors, all three product revision factors and two of the three product transition factors.

The two product operation factors are as follows.

(1) **Correctness** – includes:

- **Output correctness:** The completeness of the outputs specified (in other words, no pre-specified output is missing), the accuracy of the outputs (all system's outputs are processed correctly), the up-to-datedness of the outputs (processed information is up to date

as specified) and the availability of the outputs (reaction times do not exceed the specified maximum values, especially in online and real-time applications).

- **Documentation correctness.** The quality of documentation: its completeness, accuracy, documentation style and structure. Documentation formats include hard copy and computer files – printed manuals as well as electronic "help" files – whereas its scope encompasses installation manuals, user manuals and programmer manuals.
- **Coding qualification.** Compliance with coding instructions, especially those that limit and reduce code complexity and define standard coding style.

(2) **Reliability**. The frequency of system failures as well as recovery times.

The three product revision factors are as follows.

(1) **Maintainability.** These requirements are fulfilled first and foremost by following the software structure and style requirements and by implementing programmer documentation requirements.
(2) **Flexibility**. Achieved by appropriate planning and design, features that provide an application space much wider than necessary for the current user population. In practice, this means that room is left for future functional improvements.
(3) **Testability.** Testability includes the availability of system diagnostics to be applied by the user as well as failure diagnostics to be applied by the support center or the maintenance staff at the user's site.

Lastly, the two product transition factors are as follows.

(1) **Portability.** The software's potential application in different hardware and operating system environments, including the activities that enable those applications.
(2) **Interoperability.** The package's capacity to interface with other packages and computerized equipment. High interoperability is achieved by providing capacity to meet known interfacing standards and matching the interfacing applied by leading manufacturers of equipment and software.

To sum up – the efforts to assure the quality of maintenance services should begin early in the software development phase, when each of the quality factors reviewed above is specified in the project requirements and again later, when integrated in the project design.

The above seven factors and their distinctive impact on the various software maintenance components are presented in Table 11.1.

Table 11.1: Quality factors: impacts on software maintenance components

Quality factor	Quality sub-factors	Software maintenance components		
		Corrective	Adaptive	Functionality improvement
Correctness	Output correctness	High		
	Documentation correctness	High	High	High
	Coding qualification	High	High	High
Reliability		High		
Maintainability		High	High	High
Flexibility			High	
Testability		High		
Portability			High	
Interoperability			High	

11.2.2 Foundation 2: maintenance policy

The main maintenance policy components that affect the success of software maintenance are the version development and change policies to be applied during the software's life cycle.

Version development policy

This policy relates mainly to the question of how many versions of the software should be operative simultaneously. While it is clear that this is not an issue for custom-made software that serves one organization, the number of versions becomes a major issue for COTS software packages that are planned to serve a large variety of customers. The version development policy for the latter can take a "sequential" or "tree" form. When adopting a *sequential* version policy, only one version is made available to the entire customer population. This version includes a profusion of applications that exhibit high redundancy, an attribute that enables the software to serve the needs of all customers. The software must be revised periodically but once a new version is completed, it replaces the version currently used by the entire user population.

When adopting a *tree* version policy, the software maintenance team supports marketing efforts by developing a specialized, targeted version for groups of customers or a major customer once it is requested. A new version is inaugurated by adding special applications or omitting applications, depending on what is relevant to customer needs. The versions vary in complexity and level of application – targeted industry-oriented applications and so forth. If this policy is adopted, the software package can evolve into a multi-version package after several years of service, meaning it will resemble a tree, with several main branches and numerous secondary branches, each branch representing a version with specialized revisions. As opposed to sequential version software, maintenance and management of tree version software is much

more difficult and time-consuming. Considering these deficiencies, software development organizations try to apply a *limited tree* version policy, which allows only a small number of software versions to be developed.

Example: After a few years of application, *Inventory Perfect*, an inventory management package developed according to the tree policy, has evolved into a seven-version software package with these main branches: Pharmacies, Electronics, Hospitals, Bookstores, Supermarkets, Garages–Auto Repairs, and Chemical Plants. Each of the branches includes four or five secondary branches that vary by number of software modules, level of implementation or specific customer-oriented applications. For example, the Bookstores version has the following five secondary branches (versions): bookstore chains, single bookstores, advanced management bookstores, and special versions for the LP bookstore chain and for CUCB (City University Campus Bookstores). The software maintenance team tends to a total of 30 different versions of the software package simultaneously, with each version revised periodically according to customer requests and the team's technical innovations.

The daily experience of the maintenance team therefore includes overcoming hardships created by the version structure of the package that go beyond those related to the software itself:

- Faulty corrections caused by inadequate identification of the module structure of the current version used by the specific customer.
- Faulty corrections caused by incorrect replacement of a faulty module by a module of another version that later proved to be inadequate for integration into the customer's package version.
- Efforts invested to convince customers to update their software package by adding newly developed modules or replacing current module versions by a new version. Following successful efforts to persuade customers to update their software package, the problems and failures incurred when attempting to integrate newly developed modules or to replace current with advanced versions of the modules.

The head of the maintenance team has often mentioned that she envies her colleague, the head of *Inventory Star*'s maintenance team, who had insisted that the software package developed by his firm was to offer only one comprehensive version for all customers.

It is clear that the sequential policy adopted by *Inventory Star* requires much less maintenance; as only one version has to be maintained, it is much easier to maintain its quality level.

Change policy

Change policy refers to the method of examining each change request and the criteria used for its approval. It is clear that a permissive policy, whether implemented by the CCB (the Change Control Board) or any other body authorized to approve changes, contributes to an often-unjustified increase

in the change task load. A balanced policy, one that requires thorough examination of change requests, is to be preferred as it allows staff to focus on the most important and beneficial changes, as well as those that they will be able to perform within a reasonable time and according to the required quality standards. This policy will, of course, culminate in the approval of only a small proportion of change requests.

For more about change control, see Chapter18.

11.3 Pre-maintenance software quality components

Like pre-project SQA components, the pre-maintenance SQA activities to be completed prior to initiating the required maintenance services are of utmost importance. These entail:

- Maintenance contract review
- Maintenance plan construction.

11.3.1 Maintenance contract review

When considering the maintenance contract, a broad perspective should be embraced. More than anything else, decisions are required about the categories of services to be contracted. These decisions depend on the type of customers served: customers for whom a custom-made package has been developed, customers who purchased a COTS software package, and internal customers. So, before commencing to supply software maintenance services to any of these customers, an adequate maintenance contract should be finalized that sets down the total range of maintenance obligations according to the relevant conditions.

Implementation tip

Maintenance services to internal customers are often not contracted. In a typical situation, some of the services provided during the running-in period are continued, with no one bothering to determine the binding obligations for continuation of these services. In such situations, dissatisfaction is expected on both sides: the internal customers feel that they need to ask for favors instead of receiving the regular service that they deserve, whereas the development team eventually experiences requests to perform maintenance tasks as intrusions once they have begun work on another project.

To prevent these tensions, an "internal service contract" should be written. In this document, the services to be provided by the internal maintenance team to the internal customer are clearly defined. By eliminating most of the misunderstanding related to these vital services, such a contract can serve as the basis for satisfactory maintenance to internal customers.

The maintenance contract review activities include proposal draft reviews as well as contract draft reviews. Naturally, the objectives and implementation of maintenance contract reviews follow the lines of pre-project contract reviews (see Chapter 5). We next list the major objectives of software maintenance contract reviews.

(1) **Customer requirements clarification**

The following issues deserve special attention:

- Type of corrective maintenance services required: list of remote services and on-site services to be provided, hours of service, response time, etc.
- Size of the user population and the types of applications to be used.
- Location of users, especially of long-distance (or overseas) sites and the types of applications installed in each.
- Adaptive and functionality improvement maintenance to be provided and procedures for submission of requests for service as well as proposing and approving performance of these services.

(2) **Review of alternative approaches to maintenance provision**

The following options deserve special consideration:

- Subcontracting for sites or type of service
- Performance of some services by the customer himself with support from supplier's maintenance team.

(3) **Review of estimates of required maintenance resources**

First, these estimates should be examined on the basis of the required maintenance services, clarified by the proposal team. Then, the company's capacity to meet its commitments with respect to professional competence as well as availability of maintenance teams should be analyzed.

(4) **Review of maintenance services to be provided by subcontractors and/or the customer**

This review refers to the definition of the services provided by each participant, payments to subcontractors, quality assurance and follow-up procedures to be applied.

(5) **Review of maintenance costs estimates**

These estimates should be reviewed on the basis of required resources.

11.3.2 Maintenance plan

Maintenance plans should be prepared for all customers, external and internal. The plan should provide the framework within which maintenance provision is organized. Hence, as anticipated, the maintenance and development plans (see Chapter 6) are based on similar concepts.

The plan includes the following:

(1) **A list of the contracted maintenance services**

- The internal and external customers, the number of users, the locations of each customer site.

- The characteristics of corrective maintenance services (remote and on site).
- The obligations for adaptive and functional improving maintenance service provision for each customer.

(2) **A description of the maintenance team's organization**

The maintenance team organization plan focuses on manpower requirements, which should be carefully considered according to these criteria:

- The number of required team members. If services are to be provided from several facilities, the team requirement for each facility.
- The required qualifications for team members according to the maintenance tasks, including acquaintance with the software package(s) to be maintained.
- Organizational structure of the maintenance teams, including names of team leaders.
- Definition of tasks (responsibility for customers, types of applications, etc.) for each team.
- Training needs.

Implementation tip

In determining the maintenance team and its organization, one should consider preparing for peak demand for corrective maintenance services. The support in peak situations can be based on temporary use of development and other maintenance teams located at the same or other facilities. It should be emphasized that effective peak-load support is based on pre-planning, which includes training. Maintenance teams require regular training for these tasks; on-the-spot improvised solutions may prove to be harmful rather than helpful.

(3) **A list of maintenance facilities**

Maintenance facilities – the infrastructure that makes it possible to provide services – include:

- The maintenance support center with its installed hardware and communication equipment to provide user support and software correction services.
- A documentation center containing a complete set of documents (in printed or electronic format):
  - The software documentation, including the development documentation
  - The service contracts
  - The software configurations for each customer and versions of the software packages installed at each site, provided by configuration management
  - The maintenance history records for each user and customer.

(4) **A list of identified maintenance service risks**

Maintenance service risk relates to situations where failure to provide adequate maintenance is anticipated. These risks include:

- Staff shortages, whether throughout the organization's maintenance services, in a specific maintenance support center or for a specific application.
- Inadequate qualifications or acquaintance with part of the relevant software packages for performing user support services and/or corrective maintenance tasks.
- Insufficient team members qualified to perform functional improvement as well as adaptive tasks, in cases where a customer places an order of a significant size.

(5) **A list of required software maintenance procedures and controls**

Most of the required procedures refer to the processes implemented by the corrective maintenance teams and by the user support center. These procedures typically deal with:

- Handling customers' applications
- Handling a software failure report
- Periodic reporting and follow-up of user support services
- Periodic reporting and follow-up of corrective maintenance services
- Training and certification of maintenance team members.

For more about software quality procedures, see Chapter 14.

(6) **The software maintenance budget**

The estimates used in the corrective maintenance budget are based on the manpower organization plan, required facilities and investments needed to establish these facilities, team training needs and other tasks. They can be prepared once the manpower, facilities and procedures have been defined. Estimates for adaptive and functionality improvement maintenance tasks are prepared according to the expected workload induced by the changes and improvements to be carried out.

11.4 Maintenance software quality assurance tools

A great variety of software quality assurance tools are used throughout the operational period of the software life cycle. The specific nature of each component of software maintenance – corrective maintenance, adaptive maintenance and functionality improvement maintenance – demands that different sets of SQA tools be used for each. Furthermore, the operational period of the software typically makes intensive use of infrastructure SQA tools and managerial control tools more probable.

Some indication of the extent of resources invested in SQA during maintenance has been prepared by Perry (1995). In a survey he carried out in November 1994, the participants reported that based on their experience,

31% of their maintenance schedules were dedicated to quality assurance (reviews and testing tasks).

The next sections are dedicated to the following subjects:

- SQA tools for corrective maintenance
- SQA tools for functionality improvement maintenance
- SQA infrastructure tools for software maintenance
- SQA tools for managerial control of software maintenance.

11.4.1 SQA tools for corrective maintenance

Corrective maintenance activities entail primarily (a) user support services and (b) software corrections (bug repairs). *User support services* deal with cases of software code and documentation failures, incomplete or vague documentation; they may also involve instruction of users who have insufficient knowledge of the software or fail to use the available documentation. *Software correction services* – bug repairs and documentation corrections – are called for in cases of software failures, and are typically provided during the initial period of operation (despite the efforts invested in testing) and continue to be required, though in lower frequency. As the two types of service are inherently different, distinctive sets of quality assurance tools are used irrespective of the shared focuses on quality of service. Nonetheless, in many cases the same team performs both types of corrective maintenance.

In addition to infrastructure and management control SQA tools (discussed later in this section), most bug repair tasks require the use of *mini life cycle* SQA tools, mainly mini-testing. Mini-testing procedures are required to handle *repair patch* (small-scale) tasks, characterized by a small number of coding line changes together with intense pressure to complete the corrections rapidly. The implications of delayed repair are such that an abridged – mini – form of testing is often employed. However, use of these mini testing tools should be retained to avoid compromise situations of no testing.

In order to assure "mini testing" quality, these guidelines should be adhered to:

- Testing is to be performed by a qualified tester, not by the programmer who carried out the repair.
- A testing procedure document (in most cases 2–3 pages long) should be prepared. Included in the document are a description of the anticipated effects of the repair, the scope of corrections and a list of test cases to be activated. A re-testing procedure document, similar to the testing procedure document, should be also be prepared to handle testing of repairs of errors detected in previous tests.
- A test report fully documenting the errors detected in each stage of testing and re-testing should be completed.
- The head of the testing team is to review the testing documentation for the scope of corrections, the adequacy of the test cases and the testing

results. Responsibility for approval of the repaired software for operational (sometimes termed "production") use rests with the team's head.

- For repairs considered "simple and trivial", especially for those performed at the customer's site, mini-testing may be avoided.

Subcontracting (outsourcing) maintenance services, especially user support services, has become quite common whenever it is too troublesome or uneconomic for the maintenance contractor to directly provide these services. The main tool to assure the quality of the subcontractor's maintenance services and pave the way for smooth relations is the contractor–subcontractor contract. The SQA tools integrated into the contract focus on:

- Procedures for handling a specified range of maintenance calls.
- Full documentation of the service procedures.
- Availability of records documenting professional certification of the subcontractor's maintenance team members, for contractor review.
- Authorization for the contractor to carry out periodic review of the maintenance services as well as customer satisfaction surveys.
- Quality-related conditions requiring imposition of penalties and termination of the subcontracting contract in extreme cases.

Once maintenance becomes operative, the contractor should regularly conduct the agreed-upon reviews of maintenance service and customer satisfaction surveys.

Implementation tip

Many of the bitter failures experienced with software maintenance contracts are due to subcontracting. Failures often result from lax control over the subcontractor's performance, not from the absence of software quality assurance clauses from the contract. The reasons for subcontracting, such as a shortage of maintenance professionals at the remotely located customer's site that consumes the subcontracted services, may induce faulty control over the subcontractor's services. In other words, successful subcontracting requires adequate organization and procedures to implement proper control over performance.

More about subcontracting may be found in Chapter. 12.

11.4.2 SQA tools for functionality improvement maintenance

Due to the similarity of functionality improvement maintenance tasks to software development project tasks, project life cycle tools (reviews and testing) are regularly applied for functionality improvement maintenance. These same tools are also regularly implemented for large-scale adaptive mainte-

chapter 12

Assuring the quality of external participants' contributions

Chapter outline

Evidence for the importance of assuring the quality of external participants' contributions is found in the ISO 9000-3 Standard (see ISO, 1997, Sec. 4.6 and ISO/IEC, 2001, Sec. 7.4), IEEE Std 1062 (IEEE, 1998) and the software quality assurance literature (see Basili and Boehm, 2001; Oskarsson and Glass, 1996).

After completing this chapter, you will be able to:

- Explain the difference between contractors and external participants.
- List the types of external participants, and explain the benefits they provide to the contractor.
- Describe the risks for the contractor associated with turning to external participants.
- List the SQA tools appropriate for use with external participants and add short statements regarding the risks they help to eliminate or reduce.

12.1 Introduction: the HealthSoft case

The RedAid Health Insurance tender presented a real challenge for HealthSoft, a software house that specialized in hospital and pharmacy software. The tender's main item was an integrative nationwide system for online handling of fees charged by hospitals for services, by pharmacies for prescriptions, by physicians for clinic visits and by medical laboratories for tests. The tender also included a comprehensive patient's personal health information service to be made available through the Internet. The customer's Management Information System (MIS) Department was to develop the home office modules, based on the existing software. In addition, the MIS Department will purchase and install the hardware and communication equipment according to the contractor's specifications, see to the computerized interfacing agreements required with RedAid's suppliers of health services, and instruct RedAid personnel in the new system's operation. All the systems were to be under tight security, with high reliability requisite for all the components. The system was to become fully operative not later than 13 months after signing the contract, with the contractor fully responsible for the quality and timely completion of all system parts.

Already at the beginning of preparing the RedAid tender proposal, the HealthSoft tender team realized that they needed the professional support of companies that specialize in software security and data communication. The size of the anticipated programming load led the team to decide that a subcontractor would carry out 60%–70% of the programming load. Cape-Code, a very small software house located in a nearby suburb, was chosen as the programming subcontractor on the basis of the lowest price proposed. Some "breathing space" when preparing the proposal was obtained when the team discovered that the new enhanced Medal Software's product Version 5E of the widely used Medal Version 5, a laboratory accounting software program, included important new modules. These new modules for online external authorization of patient credit and for the preparation of monthly accounts for organizational customers like RedAid suited the tender requirements. Medal's developers had stressed the wide variety of their package's interfacing capabilities, which were touted as suited to almost any requirements. The integration of Medal's 5E version into the proposed software solved one of

the remaining difficulties hampering completion of the proposal and enabled substantial reduction of development costs. Finally, HealthSoft signed agreements with all the potential external participants – Lion Securities, Comcom and Cape-Code, subcontractors for security, communication and programming, respectively – that framed its responsibility for financial issues as well as coordination between the various organizations.

The day HealthSoft was announced winner of the tender was one of satisfaction and joy for the company. Within a few days, all the project teams were working "at full speed". Monthly coordination meetings were conducted regularly. The subcontractors reported satisfactory progress according to the project schedule. The first signs of alert appeared in the tenth meeting. Comcom, the communication subcontractor, reported that some of RedAid's major suppliers had refused to supply the information needed for planning the communication equipment to be installed on their premises as they had not reached an agreement with RedAid on the issue. As expected, Lion Securities, the security subcontractor, faced similar difficulties. Both subcontractors declared that even if full cooperation was to be achieved within a week, a one-month delay in completion of the project was inevitable. Yet, Cape-Code people continued to express their satisfaction with the progress of the development tasks they had undertaken. The next coordination meeting was a special meeting, called after only two weeks, to discuss the severe delays that had appeared in Cape-Code's schedule. The delays had been discovered by a HealthSoft team when it tried to coordinate a planned integration test. At this late stage, HealthSoft found out that Cape-Code had subcontracted the development task to another small software house. It became clear that all the previous calming reports had not been based on actual information; they were fabrications, meant to satisfy HealthSoft people (and ensure regular income to Cape-Code).

Integration tests of the Cape-Code modules, begun 10 weeks behind schedule, identified many more faults – of all kinds – than anticipated. Correction time required exceeded that planned. About the same time, the team assigned to integrate the Medal Version 5E software into the system realized that the enhanced version was not operative for all new modules, particularly the online external authorization of patients' credit status. In addition, the interfacing trials with other system modules failed. Medal Software assigned a special team to complete the development of the missing module parts and perform the necessary corrections. Though their efforts were visible, successful completion of the software integration tests was accomplished almost 20 weeks behind schedule.

The system test started 19 weeks behind schedule, with the same severity of quality problems that had been observed at the integration phase. Finally, about five months late, it became possible to install the hardware and software equipment in RedAid's main office and at its suppliers' sites.

The three-week conversion phase of the project, begun 23 weeks behind schedule, was, surprisingly, a great success, with no major faults discovered and immediate repair of all faults that were revealed. However, the implementation

phase was a colossal failure: only one-third of the staff listed for training actually participated in the instruction courses, and the majority of those participating displayed insufficient preliminary knowledge of the new systems. Success with supplier personnel was even lower. Only eight weeks later could regular operation of the system begin, but with only about half of RedAid's suppliers integrated into the new system.

The project, a frustrating one for all who participated, ended with a series of court claims. RedAid sued HealthSoft, and HealthSoft sued RedAid, CapeCode and Medal Software, the developers of the Medal software package. Lion Securities and Comcom decided not to sue HealthSoft – despite the extra costs they had incurred following RedAid's lack of cooperation and the subsequent obstacles raised to efficient performance of their parts in the project – in expectation of continuing cooperation with HealthSoft on future projects. The trials lasted for years. The only consolation was that the new software, once in operation, was a great success, with many of RedAid's management admitting that the system worked well beyond their expectations.

You may ask yourself:

- Could the final gratifying results have been achieved without the "mess" experienced during the course of the project?
- Could they have been achieved without the major losses faced by all the participants?
- Was the HealthSoft method of choosing subcontractors satisfactory?
- Was the method of purchasing COTS software appropriate?
- Was the method of controlling the implementation of the customer's contribution to the project adequate?
- Was HealthSoft's control over its external participants adequate?

Whatever your responses to the specific questions, we can readily claim that had HealthSoft properly implemented SQA activities, problems like those described could have been avoided. Prevention of such troubles is the subject of this chapter.

12.2 Types of external participants

The partners to a software development project – the organization that is interested in the software system (the "customer") and the organization that undertakes to carry out the development (the "contractor") – are nowadays often not the only participants in the project. The external participants involved in a software development project contribute to the project but are not contractors, nor are they the contractor's partners. Their contributions to the project are structured through agreements with the contractor (subcontractors and suppliers of COTS software) or through those clauses of the project con-

tract that state what parts of the project will be performed by the customer himself. The larger and more complex the project, the greater the likelihood that external participants will be required, and the larger the proportion of work to be transmitted or parceled out. The motivation for turning to external participants rests on several factors, ranging from the economic to the technical and to personnel-related interests, and reflects a growing trend in the allocation of the work involved in completing complex projects.

External participants can be classified into three main groups:

(1) **Subcontractors** (currently called "outsourcing" organizations) that undertake to carry out parts of a project, small or large, according to circumstances. Subcontractors usually offer the contractor at least one of the following benefits: staff availability, special expertise or low prices.

(2) **Suppliers of COTS software and reused software modules.** The advantages of integrating these ready elements are obvious, ranging from timetable and cost reductions to quality. One expects that integration of these ready-for-use elements will achieve savings in development resources, a shorter timetable and higher quality software. Software of higher quality is expected as these components have already been tested and corrected by the developers as well as corrected according to the faults identified by previous customers. The characteristics of COTS software and quality problems involved in their use are discussed by Basili and Boehm (2001).

(3) **The customer themselves as participant in performing the project.** It is quite common for a customer to perform parts of the project: to apply the customers' special expertise, respond to commercial or other security needs, keep internal development staff occupied, prevent future maintenance problems and so forth. This situation does have drawbacks in terms of the customer–supplier relationship necessary for successful performance of a project, but they are overweighed by the inputs the customer makes. Hence, the inevitability of this situation has become a standard element of many software development projects and contractual relations.

Typical contracting structures of projects are presented in Figure 12.1.

12.3 Risks and benefits of introducing external participants

The main risks to project quality associated with introducing external participants within the framework of the project are as follows.

(1) **Delays in completion of the project.** In those cases where external participants are late in supplying their parts to the software system, the project as a whole will be delayed. These delays are typical for subcontractors'

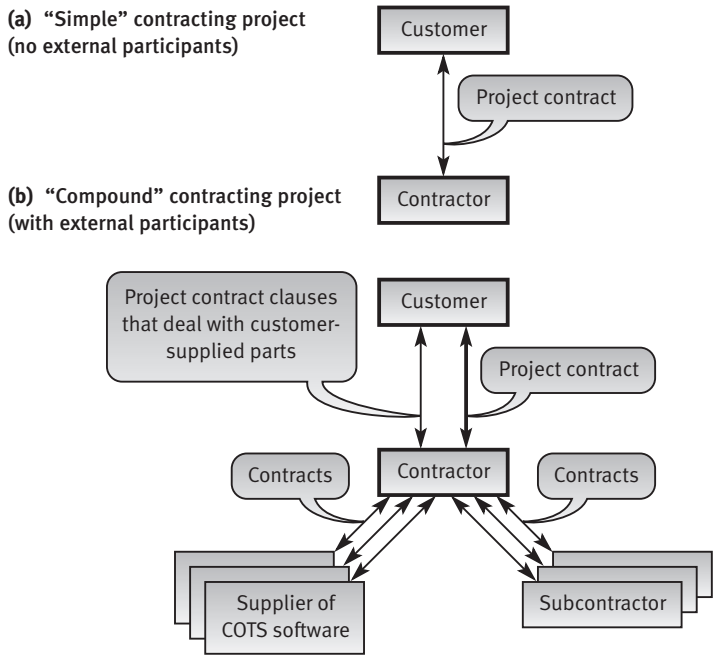


Figure 12.1: Software development projects: typical contracting structures

parts and customers' parts but less so for COTS software suppliers. In many cases the control over subcontractors' and the customers' software development obligations is loose, a situation that causes tardy recognition of expected delays and leaves no time for the changes and reorganization necessary to cope with the delays and to limit their negative effects on the project.

Implementation tip

Purchasing a software package for integration into a newly developed software system usually entails substantial savings of development resources, including budgeted funds. This is especially true when the relevant software has been tested and currently serves a substantial population of users. In some cases the contractor is persuaded to purchase a new, supposedly advanced version of an accepted software package, soon to be put on the market and touted as better suited to his/her requirements. However, it has become common to discover just a week or two later that the version's release is (unexpectedly) delayed – and repeatedly. More thorough investigation into the status of the new version, including requests for information from customers, may also reveal that vital parts – for instance, development of equipment and software interfaces or an advanced application – have been shifted to a later stage.

(2) **Low quality of project parts supplied by external participants.** Quality problems can be classified as (a) defects: a higher than expected number of defects, often more severe than expected; and (b) non-standard coding and documentation: violations of style and structure in instructions and procedures (supposedly stipulated in any contract). Low quality and non-standard software are expected to cause difficulties in the testing phase and later in the maintenance phase. The extra time required to test and correct low-quality software can cause project delays even in cases when external participants complete their tasks on time.

(3) **Future maintenance difficulties.** The fact that several organizations take part in development but only one of them, the contractor, is directly responsible for the project creates two possibly difficult maintenance situations:

   (a) One organization, most probably the contractor, is responsible for maintenance of the whole project, the arrangement commonly stipulated in the tender itself. The contractor may then be faced with incomplete and/or non-standard coding and documentation supplied by the external participants, causing lower-quality maintenance service delivered by the maintenance team and higher costs to the contractor.

   (b) Maintenance services are supplied by more than one organization, possibly the subcontractors, suppliers of COTS software and occasionally the customer's software development department. Each of these bodies takes limited responsibility, a situation that may force the customer to search for the body responsible for a specific software failure once discovered.

   Damages caused by software failures are expected to grow in "multi-maintainer" situations. Neither of these situations contributes to good and reliable maintenance unless adequate measures are taken in advance, during the project's development and maintenance planning phases.

(4) **Loss of control over project parts.** Whether intentionally or not, the control of software development by external bodies may produce an unrealistically optimistic picture of the project's status. Communication with external participants' teams may be interrupted for several weeks, a situation that prevents assessment of the project's progress. As a result, alerts about development difficulties, staff shortages and other problems reach the contractor belatedly. The possibilities for timely solution of the difficulties – whether by adaptations or other suitable changes – are thereby often drastically reduced.

Before entering into these agreements, the contractor must consider the associated benefits and risks of introducing external participants in a project. These are summarized in Frame 12.1.

Frame 12.1 Introduction of external participants: benefits and risks

Benefits

For the contractor:

1. Budget reductions
2. Remedy of professional staff shortages
3. Shorter project schedule
4. Acquisition of expertise in specialized areas

For the customer (as external participant):

1. Protecting the customer's commercial secrets
2. Provision of employment to internal software development department
3. Acquisition of project know-how for self-supplied maintenance
4. Project cost reductions

Risks

For the contractor and the customer:

1. Delayed completion of the project caused by delays in completion of parts supplied by external participants
2. Low quality of parts supplied by external participants
3. Increased probability of difficulties in maintaining parts supplied by external participants
4. Loss of control over development of specific project parts

12.4 Assuring the quality of external participants' contributions: objectives

What are the objectives to be obtained by application of SQA tools in the case of parts supplied by external participants? These objectives can be derived directly from the risks listed in Frame 12.1:

(1) To prevent delays in task completion and to ensure early alert of anticipated delays.
(2) To assure acceptable quality levels of the parts developed and receive early warnings of breaches of quality requirements.
(3) To assure adequate documentation to serve the maintenance team.
(4) To assure continuous, comprehensive and reliable control over external participants' performance.

12.5 SQA tools for assuring the quality of external participants' contributions

We can expect external participants to operate their own SQA systems that include the tools necessary for achieving acceptable quality levels for their own software products and services. The tools mentioned here are those that contractors can apply *vis-à-vis* their external participants. For this purpose, the issues of quality and timetable are the most important to be addressed.

The main SQA tools to be applied before and during incorporation of external participants in a software development project are listed in Frame 12.2.

Frame 12.2 SQA tools applied to external participants in a software development project

- Requirements document reviews
- Evaluation of choice criteria regarding external participants
- Establishment of project coordination and joint control committee
- Participation in design reviews
- Participation in software testing
- Formulation of special procedures
- Certification of supplier's team leaders and members
- Preparation of progress reports of development activities
- Review of deliverables (documents) and acceptance tests

12.5.1 Requirements document reviews

Requirements documents provide the formal basis for the contracts signed between the contractor and subcontractors as well as for the contract clauses dealing with the customer's obligations to carry out parts of the project. The requirements document is vital for the examination of proposals presented by suppliers of COTS software and the subsequent negotiations regarding their participation. Hence, review of the requirements documents to be presented to external participants is meant to assure their correctness and completeness. The principles guiding the review are similar to those of contract reviews, adjusted to the different role of the contractor in this case – as the customer.

In general, the requirements documents presented by contractors to external participants should be correlated with the customer's requirements. The main issues to be dealt with in a requirements document are presented in Table 12.1.

Table 12.1: Requirements list presented to external participants

Requirements type	Requirements subject
Software functionality	(1) Functional requirements (related to the customer's requirements) (2) Interfaces between the external participant's part and other parts of the project (3) Performance, availability, usability and reliability (related to the customer's requirements) (4) Maintenance services that will be required
Formal and staff	(1) Required qualifications of team leaders and members, including certification where applicable (2) Establishment of coordination and joint control committee including procedures for handling complaints and problems (3) List of documents to be delivered by the external participant (4) Criteria for completion of external participant's part (5) Financial arrangements, including conditions for bonuses and penalties
SQA	(1) Requirements regarding participation in the external participant's design reviews (2) Requirements regarding participation in the external participant's software testing

Implementation tip

One of the main surprises encountered by contractors is the revelation that the subcontractor – without any authorization or prior consent – has subcontracted his task to another company. Whatever the reason or justification for this step, it usually leads to a loss of contractor control over project quality, with the subsequent delays and non-compliance with quality requirements.

Contract clauses dealing with these issues are often inadequate to prevent such behavior. Improved prospects for elimination of such behavior can be achieved only by combining stringent contractual clauses with strict implementation controls.

12.5.2 Choice of external participants

While it is clear that the case of customer participation in a project is difficult if not impossible to circumscribe or prevent, a good degree of choice is available with respect to the other external participants: subcontractors and suppliers of COTS software. General quality assurance procedures, with the appropriate adaptations, can be applied in this situation as well. Any choice of external participants requires collection of information about the candidates, their products and team qualification, and evaluation of that information.

Collection of information

The main tools that support choice are:

- Contractor's information about suppliers and subcontractors based on previous experience with their services
- Auditing the supplier's or subcontractor's quality assurance system
- Survey of opinions regarding the external participants from outside sources.

(1) **Use of contractor's internal information about suppliers and subcontractors.** An external participant (subcontractor or supplier) file, that records past performance, is the main source of information for the contractor. Such an information system is based on cumulative experience with tasks performed by the subcontractor or supplier of COTS software, as well as on information gathered for evaluation of their past proposals. Implementation of this tool requires systematic reporting, based on SQA procedures, by the departments involved:

- Teams of committees that evaluate suppliers' proposals
- User representatives and coordination committee members who are responsible for project follow-up
- "Regular" users who have identified software faults or have gained experience with the supplier's products and maintenance service.

Implementation tip

Two issues impinging on the adequacy of a "Suppliers' File" should be considered:

(a) Individuals evaluating a proposal like to receive full documentation on the organization's past experience with a prospective subcontractor/supplier together with information gathered in the past from various outside sources. Yet, these same individuals are likely to neglect preparing records related to their own experience with an external participant.

(b) Difficulties often result from unstructured reporting to the Suppliers' File. If the information is not properly structured, evaluation and comparison of suppliers become taxing, if not impossible.

The answer to these difficulties frequently lies in the procedures applied and forms used. Procedures that define who should report what and in which situations can limit the reporting burden. A structured reporting form, supported by unstructured descriptions, can be helpful in responding to both issues.

(2) **Auditing the supplier's quality system.** Auditing the supplier's SQA system is often encouraged by the suppliers themselves in an effort to promote acceptance of their proposals. In some cases such an audit is part of the tender requirements. The auditors should take care that the audited features are relevant to the project in its content, magnitude and complexity. Another issue to be considered is the demonstration project and team, which are usually chosen by the supplier. The preferred route is, of course, for the auditors to randomly choose the project and team from a list of relevant projects and teams. This approach is, however, rarely adopted due to objections voiced openly or implicitly by subcontractor and/or supplier.

(3) **Opinions of regular users of the supplier's products.** Opinions can be gathered from internal units that used the supplier's services in the past, from other organizations that have experienced the supplier's services in the past, from professional organizations that certified the supplier as qualified to specialize in the field, and from firms that have had professional dealings with the potential subcontractor or supplier. The purpose of this step is also to ascertain reliability, among other variables that may affect contractual relations.

Systematic evaluation of the suppliers

Evaluation and comparison of potential external participants should be carried out according to procedures adequate to their purpose. Among the factors set down in the procedure are designation of the evaluation committee or responsible manager and the process of evaluation, including the method for defining the relative importance attached to each item and source of information.

12.5.3 The project coordination and joint control committee

The scope of this committee's activities and responsibilities varies in direct relation to the part the external participant will play in the project. Naturally, these will be rather limited in the case of purchased COTS software or reused software in cases where the required supplier's support is minimal and no maintenance is required. Alternatively, substantial coordination and progress control are demanded when subcontractors are to carry out major parts of the project.

The committee's main activities are:

- Confirmation of the project timetable and milestones
- Follow-up according to project progress reports submitted to the committee
- Meeting with team leaders and others in the field in severe situations
- Making decisions about solutions to timetable and resource shortage problems arising during the project that have been identified during follow-up

- Making decisions regarding solution of problems identified in design reviews and software tests
- Solving disagreements about contract implementation.

Application of the specific SQA procedure that regulates follow-up of external participants' work activities can be of great help.

12.5.4 Participation in design reviews

The extent to which contractors' participation is required in subcontractors' design reviews or customers' reviews of other development activities depends on the nature of the project parts provided by the external participants. When the contractor participates, we can expect him or her to act as a full member of the review. In other words, he or she will read and review the documents before the team's meeting and participate in the team's discussions as well as in the decision taken at the end of the review.

12.5.5 Participation in software testing

Participation in software testing, when required, should include all the stages of the testing process: design reviews of the planning and design of the tests, reviews of the test results, follow-up meetings for the corrections and regression testing. That is, the character of participation in the testing process is sufficiently comprehensive to enable the contractor's representative to intervene, if necessary, to obtain assurance of the quality demanded of the supplied software and the expected timetable for completion of the testing (and correction) process.

12.5.6 Specialized procedures

The specialized procedures that regulate SQA activities within the context of contractual relations with external organizations (i.e., organizations that are not partners in the project contract) have already been mentioned in this chapter. These special procedures are usually adaptations of procedures applied in projects that the organization has carried out. Here, these procedures are mentioned in greater detail. Usually, they are supported by templates, checklists and forms that attach extra value to the fundamental procedures. The main objectives of specialized procedures are:

- Preparation of requirements documents for external participants
- Choice of a subcontractor or supplier of COTS software
- Audit of the subcontractor's SQA system
- The Suppliers' File, its sources of information and mode of operation

- Appointment of the coordination and joint control committee for project parts to be carried out by external participants and preparation of instructions for its operation
- Progress reporting requirements for project parts carried out by external participants.

12.5.7 Certification of external participants' team leaders and other staff

Qualification and certification of the external participants' team leaders and other staff are intended to ensure an acceptable level of professional work *as required by the project or the customer.* This requirement is not to be belittled, for the quality of staff is the heart of any contractual relationship. The SQA activities required here are:

- Qualification and certification of staff should be listed as a contractual requirement
- Implementation of these clauses is to be confirmed by the contractor at the outset of the work
- Changes and replacement of the respective team members are to be approved by the contractor
- Implementation of these clauses by the contractor is to be periodically reviewed.

Implementation tip

Subcontractors under pressure from other projects or for other reasons frequently try to replace qualified and professional certified team members needed elsewhere with staff who are not fully qualified and/or lack certification. "Partial" violations – with the team leader or team member allocating his or her time, without approval, on more than one project – are also common. The control activities mentioned should deter the subcontractor from changing staff in this manner and help the contractor quickly identify violations should they occur.

12.5.8 Progress reports

When external participants share the project's workload, the main progress reports prepared for the coordination and joint progress control committee are as follws:

- **Follow-up of the risks identified in the project work.** This report describes the current status of the risks identified in previous reports, such as shortage of professionals having special expertise, shortage of equipment, and difficulties in development of a module. For risks still

chapter 13

CASE tools and their effect on software quality

Chapter outline

An increasing variety of specialized computerized tools (actually software packages) have been offered to software engineering departments since the early 1990s. The purpose of these tools is to make the work of development and maintenance teams more efficient and more effective. Collectively named CASE (computer-aided software engineering) tools, they offer:

- Substantial savings in resources required for software development
- Shorter time to market
- Substantial savings in resources required for maintenance
- Greater reuse due to increased standardization of the software systems
- Reduced generation of defects coupled with increased "interactive" identification of defects during development.

It is clear that this last characteristic is the one most attracting the interest of software quality analysts to CASE tools.

In light of their characteristics, CASE tools serve as a source for easing the amount of effort expended on development of increasingly complex and large software systems.

The following sections will deal with the subjects:

- What is a CASE tool?
- How can CASE tools contribute to the improved quality of software products?

- How can CASE tools contribute to the improved quality of software maintenance?
- How and to what extent can CASE tools contribute to maintaining development process timetables and keeping with budgets?

After completing this chapter, you will be able to:

- Explain the difference between "classic" and "real" CASE tools and provide examples of each.
- Explain the contribution of CASE tools to software development.
- List the main contributions of real CASE tools to product quality.
- Explain the contribution of CASE tools to software quality maintenance.

13.1 What is a CASE tool?

Frame 13.1 contains the basic definition of a CASE tool.

Frame 13.1 CASE tools – definition

CASE tools are computerized software development tools that support the developer when performing one or more phases of the software life cycle and/or support software maintenance.

The definition's generality allows compilers, interactive debugging systems, configuration management systems and automated testing systems to be considered as CASE tools. In other words, well-established computerized software development support tools (such as interactive debuggers, compilers and project progress control systems) can readily be considered *classic* CASE tools, whereas the new tools that support the developer for a succession of several development phases of a development project are referred to as real CASE tools. When referring to *real* CASE tools, it is customary to distinguish between *upper* CASE tools that support the analysis and design phases, and *lower* CASE tools that support the coding phase (where "upper" and "lower" refer to the location of these phases in the Waterfall Model – see Section 7.1), and *integrated* CASE tools that support the analysis, design and coding phases.

The main component of real CASE tools is the *repository* that stores all the information related to the project. The project information accumulates in the repository as development proceeds and is updated as changes are initiated during the development phases and maintenance stage. The repository of the previous development phase serves as a basis for the next phase. The accumulated development information stored in the repository provides support for the maintenance stage in which corrective, adaptive and functionality improvement tasks are performed. The computerized management of the repository guarantees the information's consistency and its compliance with project methodology as well as its standardization according to style and

structure procedures and work instructions. It follows that CASE tools are capable of producing full and updated project documentation at any time. Some lower CASE and integrated CASE tools can automatically generate code based entirely on the design information stored in the repository. Reverse engineering (re-engineering) tools are also considered to be real CASE tools. Based on the system's code, these tools are applied mainly for recovery and replication of (now non-existing) design documents for currently used, well-established software systems ("legacy" software). In other words, reverse engineering CASE tools operate in the opposite direction of "regular" CASE tools: instead of creating system code on the basis of design information, they automatically create complete, updated repository and design documents on the basis of system code.

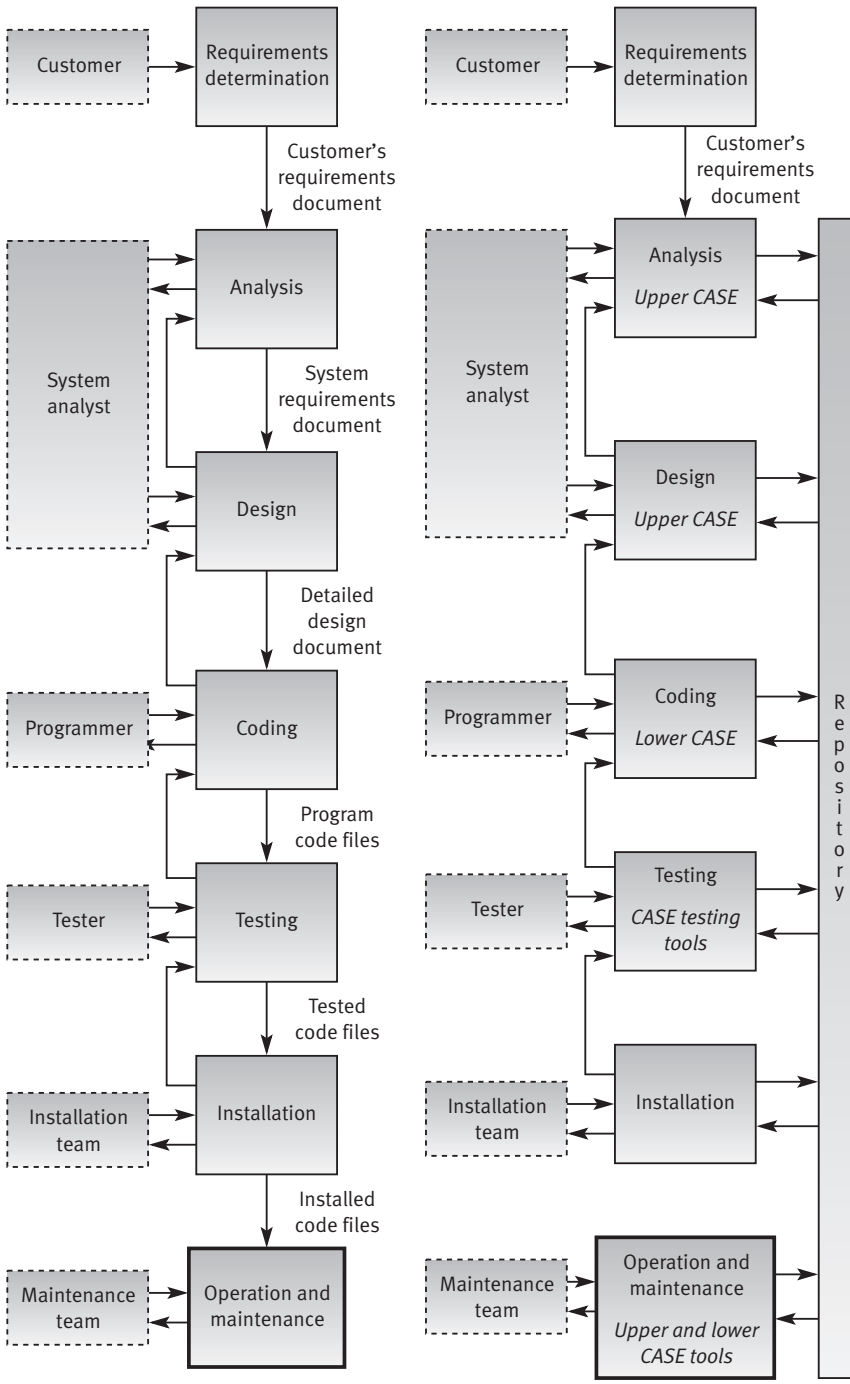
Figure 13.1 describes the application of CASE tools in the development process in comparison to the traditional development process.

The support that CASE tools provide the developer can be in one or more of the following areas, listed in Table 13.1.

Table 13.1: CASE tools and the support they provide to developers

Type of CASE tool	Support provided
Editing and diagramming	Editing text and diagrams, generating design diagrams according to repository records
Repository query	Display of parts of the design texts, charts, etc.; cross-referencing queries and requirement tracing
Automated documentation	Automatic generation of requested documentation according to updated repository records
Design support	Editing design recorded by the systems analyst and management of the data dictionary
Code editing	Compiling, interpreting or applying interactive debugging code for specific coding language or development tools
Code generation	Transformation of design records into prototypes or application software compatible with a given software development language (or development tools)
Configuration management	Management of design documents and software code versions, control of changes in design and software code*
Software testing	Automated testing, load testing and management of testing and correction records, etc.
Reverse engineering (re-engineering)	Construction of a software repository and design documents, based on code: the "legacy" software system. Once the repository of the legacy software is available, it can be updated and used to automatically generate new versions of the system. As new re-engineered software version is generated, it can be easily maintained and its documentation automatically updated
Project management and software metrics	Support progress control of software development projects by follow-up of schedules and calculation of productivity and defects metrics

* For more about configuration management, see Chapter 18.



(a) Traditional development life cycle **(b)** Real CASE tool-supported development life cycle

Figure 13.1: Traditional vs. CASE-supported development life cycle

More information about CASE tools can be found in IEEE Std 1462 (IEEE, 1998) and in the software engineering literature, particularly Pressman (2000), Sommerville (2001) and Kendall and Kendall (1999). The impact of CASE tools on software quality assurance is discussed by McManus (1999).

13.2 The contribution of CASE tools to software product quality

CASE tools contribute to software product quality by reducing the number of errors introduced in each development phase. In order to evaluate this contribution, we now examine the quality improvements anticipated for each of the nine causes of software errors listed in Section 2.3. We include classic and real CASE tools in our evaluation.

Table 13.2 lists the contributions CASE tools can make to quality.

Table 13.2: CASE tools and the quality of software products

	Extent and manner of contribution to quality	
Cause of software errors	**Classic CASE tools**	**Real CASE tools**
1. Faulty requirements definition		**Almost no contribution** Computerized examination of requirements consistency or correctness is rarely possible.
2. Client–developer communication failures		**Almost no contribution** In most cases, computerized identification of communication failures is impossible. Communication failures can be located or prevented only when a change or other information is found to be inconsistent with repository information.
3. Deliberate deviations from software requirements		**High contribution** Based on information stored in the repository, deviations from recorded requirements are identified as inconsistent and labeled as errors. Such deviations can also be identified by repository-based requirements tracing tools and cross-referenced query tools.
4. Logical design errors		**High contribution** (1) Re-engineering enables automated generation of the design of legacy systems and their recording in a repository.

Table 13.2 continued

Cause of software errors	Extent and manner of contribution to quality Classic CASE tools	Real CASE tools
4. Logical design errors		**High contribution** (2) Use of the repository is expected to identify design omissions, changes and additions inconsistent with repository records.
5. Coding errors	**Very high contribution** Application of compilers, interpreters and interactive debuggers.	**Very high contribution** Application of lower CASE tools for automated code generation achieves full consistency with the design recorded in the repository. In addition, as coding is automatic, no coding errors are expected.
6. Non-compliance with coding and documentation instructions	**Limited contribution** Use of text editors and code auditing supports the standardization of structure and style of texts and code and facilitates identification of non-compliance.	**Very high contribution** Application of lower CASE tools for automated code generation assures full compliance with documentation and coding instructions.
7. Shortcomings in the testing process	**High contribution** Automated testing tools perform full regression and automated load testing. Computerized management of testing and corrections reduces errors by improvement follow-up.	**High contribution** Application of lower CASE but especially of integrated CASE tools prevents coding errors and reduces design errors. Application of repository tools (cross-referenced queries and performance consistency checks) to corrections and changes during the development process prevent most software errors.
8. Procedural errors	**High contribution** Control of versions, revisions and software installation by means of software configuration management tools.	**Limited contribution** Use of updated and full documentation is expected to prevent many of the maintenance errors caused by incomplete and/or inaccurate documentation, especially if the design has been revised several times.
9. Documentation errors	**Limited contribution** Application of text editors only	**High contribution** Use of repository automatically generates full and updated documentation prior to each correction or change.

13.3 The contribution of CASE tools to software maintenance quality

Classic but especially real CASE tools contribute to the various types of software maintenance quality in several ways.

Corrective maintenance:

- CASE-generated full and updated documentation of the software enables easier and more reliable identification of the cause for software failure.
- Cross-referenced queries enable better identification of anticipated effects of any proposed correction.
- Correction by means of lower CASE or integrated CASE tools provides automated coding, with no expected coding errors as well as automated documentation of corrections.

Adaptive maintenance:

- Full and updated documentation of the software by CASE tools enables thorough examination of possible software package adaptations for new users and applications.

Functional improvement maintenance:

- Use of the repository enables designers to assure consistency of new applications and improvements with existing software systems.
- Cross-referenced repository queries enable better planning of changes and additions.
- Changes and additions carried out by means of lower CASE or integrated CASE tools enable automated coding, with no expected coding errors as well as automated documentation of the changes and additions.

13.4 The contribution of CASE tools to improved project management

Let us compare two projects of similar nature and magnitude: Project A is carried out by conventional methods, Project B by advanced CASE tools. The following results were obtained after comparison of the planning and implementation phases:

Method of development	Project A Conventional tools	Project B CASE tools
Planned resources (man-months)	35	20
Actual resources invested	42	27
Planned completion time (months)	15	9
Actual completion time	18	12