Chapter 1

Maintenance Issues and Related Management Approaches

This chapter covers:

- **Users' and maintainers' perceptions of software maintenance**
- **Maintenance body of knowledge and maintenance definitions**
- **Organization responsible for software maintenance**
- **Software maintenance standards**
- **Maintenance process and activities**
- **Maintenance measurement, service measurement, and benchmarking**

1.1 INTRODUCTION

When successfully completed, the software development process ends with the delivery of a software product that should meet the customer's original requirements. Once in operation, a software product either changes or evolves as defects are uncovered, operating environments change, and new user requirements surface.

This chapter introduces the basic concepts that govern software maintenance. For instance, once the software is in operation, failures occur. Failures are defined as defects that have found their way into production without being caught by the developer's testing effort. Failures are detected and reported by users and are generally repaired under the initial software warranty or through postimplementation delivery maintenance activities. In addition to the correction of the software after its original implementation, there will also be changes to be considered as users formulate new requests. Although it is generally believed that the software maintenance cycle starts on the first day of operation of the new software, it is well documented

Software Maintenance Management. By Alain April and Alain Abran

that many software predelivery activities go on behind the scenes during the development process itself. These and many other activities are part of the software maintainer's knowledge, and are the maintainer's responsibility.

1.2 ISSUES IN SOFTWARE MAINTENANCE

A number of authors have identified maintenance-related issues that affect maintenance resource management and processes and their specific tools/techniques. The issues reported in the literature vary according to the point of view of the authors who identified and described them. These issues can be categorized based on two viewpoints:

1. External: perceptions of customers (i.e., users and stakeholders)
2. Internal: perceptions of software maintenance engineers and managers

From the customers' (external) point of view, the main issues reported are high cost, slow delivery of services, and fuzziness of prioritization [internal information systems and information technology (IS/IT) priorities versus users' priorities] [Pigoski 1997]. From the maintainers' (internal) point of view, key maintenance issues include software that has been poorly designed and programmed, and a major lack of software documentation [Glass 1992]. In addition, it has been often observed that *the biggest problem in software maintenance is not technical but rather its management.* Let us examine both viewpoints in more detail.

1.2.1 Users' Perceptions of Software Maintenance Issues

It has been reported that a large portion of software life cycle costs—50 to 90%—come from software maintenance, and Boehm states that, for every dollar spent on development, two will need to be spent on maintenance [Boehm 1987].

It has also been reported that a significant part of the user's perception of the high cost of maintenance may stem from inadequate communication by maintenance managers about the type of maintenance work performed on the customer's behalf. Maintenance managers too often group enhancement and corrective work together in their management reports, statistics, and budgets, which warps costs in both maintenance budgets and reports. This aggregation maintains customer misconceptions about maintenance services and perpetuates the misleading perception that software maintenance is mainly concerned with correcting defective software.

Software maintenance has some peculiarities when compared to maintenance in other domains: whereas hardware and mechanical components deteriorate when there is no maintenance, software, by contrast, deteriorates as a result of multiple maintenance activities [Glass 1992]. Because software practices have not matured to the extent that hardware practices have, maintenance is performed by identifying

one or many components and fixing them. Software maintenance often requires rework of many, if not all, parts of the software. Multiple maintenance activities are blamed for further degrading the software's internal structure and making future maintenance activities progressively more difficult. If no strategy is put in place to control these effects, the software structure and quality continue to deteriorate. Software in this state is often referred to as legacy software.

Software maintenance is also labor intensive, with the majority of costs arising from programmers' salaries. Indeed, with hardware costs no longer representing the biggest portion of IS/IT costs, that of hiring expert programmers now ranks first (e.g., more than $1500 per day for the consulting services of a programmer specializing in ERP software language).

By contrast, users' perceptions of the costs are also influenced by service annoyances, which can sometimes have a considerable impact on user perceptions about both the quality of services and their cost. For example, production outages and errors in software-supported applications, as well as delays and high costs in implementing minor functional changes are often cited as problem areas in software maintenance.

Furthermore, for software under maintenance production system failures occur randomly, and user requests come in on an irregular basis. Without agreed-upon and mature queue-management mechanisms supported by detailed service-level agreements (SLAs), users will often get service that does not meet their real and stated priorities. When they get poor maintenance services, some users overreact and rank all requests as high priority and demand that all problems and requests be addressed at the same time. Given that production-system failures are random events, and that they need to be addressed first, such users perceive that work on their requests is not progressing as they would expect. When customers become frustrated with the slow delivery of services, some will consider developing local solutions to solve their problems, or might consider subcontracting or outsourcing maintenance work altogether.

Some of the users' perceptions of the slowness of service is partially based on a misunderstanding of the fundamental difference between the electronic/computer hardware maintenance and software maintenance domains. For instance, a software user often does not have a clear idea of what constitutes a software maintenance activity, describing, for example, how the repair of a broken printer or computer component may simply involve a modular-component replacement activity. He or she also might note that electronic equipment design, being more mature than software design, is based on a thorough "modularization" of components, which does not create side-effects during maintenance activities. Moreover, when well designed, hardware allows complete testing and diagnostics of components at the touch of a button! Many users, unfamiliar with the software domain, may perceive that software is maintained in a similar fashion. Unfortunately, software has not yet matured to this stage. There remain many challenges to the existence of modularity and autotesting in software, in particular because of its lack of maturity; furthermore, most software currently in use is more than a decade old and would not incorporate this type of architectural technology.

Is this user perception representative of the actual maintainers' workload? Are the maintainers spending most of their time correcting software bugs, or providing value-added services?

As far back as 1980, Lientz and Swanson [1980] were the first to point out that 55% of requests routed to maintenance organizations concerned new functions rather than failure correction. The proposed S3^{m}® model addresses failure corrections and new functions as well as many other maintenance-related activities. This was confirmed ten years later in another study, based on data collected over a few years and for each maintenance activity category [Abran 1991]. Similarly, Pressman [2004] also notes, based on feedback collected from maintenance engineers, that, rather than spending from 50% to 80% of their effort on correcting problems, they spend the biggest portion of their effort on software evolution, on adding user-requested functionality, and on answering all kinds of questions about the business rules of application software. Unfortunately, such value-added activities are poorly communicated and rarely reported in detail to the clients as monthly maintenance accomplishments. In summary, the users' perceptions do not necessarily map well to the actual maintainers' workload: "The more substantial portion of maintenance costs is devoted to accommodating necessary functional changes to the software in order to keep pace with changing user needs. Based on the data we had reviewed, we also noted that systems with well-structured software were more able to accommodate such changes" [Fornell 1992]. Real-time, quick-turnaround maintenance requests rarely include large enhancements done using project management techniques that may take multiple years.

Software-maintenance management should, therefore, do a better job of communicating the many maintenance activities, especially the value-added ones. To do this, it is important that management understand the maintainers' processes and services and their many challenges.

Software maintainers must set up, and better communicate, that there is a fair and efficient queue and priority process in place that manages and monitors the status of each maintenance request/event. This queue management process in software maintenance has many inputs and concurrent interrupting sources, such as: (1) operators who report system failures, (2) users who notice service degradation, (3) development project managers who require current software information and inputs in reengineering studies, and (4) customers who require urgent information. And when there is contention in requests for his or her services, the software maintainer must refer to the SLA and clearly point out the process in place to manage priorities based upon agreed upon service criteria.

1.2.2 Maintainers' Perceptions of Software Maintenance Issues

Maintainers are often confronted with a million lines of somebody else's source code. They must familiarize themselves quickly and process urgent changes without disrupting service. To make maintenance work even more challenging, the new-

ly developed software often has a number of urgent changes pending that the software developers could not include in the initial development, as they had been under pressure to deliver. In a context in which financial resources are limited, the analysis, design, and testing phases of the initial software development project are all put under pressure and are often only partially completed. Such omissions necessarily lead to dire consequences for software maintenance. Maintainers feel that they have little or no control over the quality of legacy code.

It has long been recognized that an immature software development process creates a number of problems in the final product delivered to the customer and to the maintenance organization, whereas a mature process that clearly captures and manages requirements and specifications results a stable design will lower maintenance costs. Use of immature development processes creates an initial maintenance backlog that needs to be addressed, on a priority basis, during the first months/year of operation of the new software. It is observed in industry that processing this backlog creates a large increase in the number of lines of code during the first three years of software maintenance; this is recognized as a major cause of increased maintenance costs and initial dissatisfaction of customers.

Software age also has an impact on maintenance cost when the software was built using older techniques that did not take in account modern architectural concepts, with complex structures, nonstandardized code, and poor documentation, leading to further difficulty in maintenance work. The structure of software undergoing maintenance also becomes increasingly complex as it is modified over time: as the size and complexity of the software increases, an increasing number of resources is required to maintain it.

Software maintenance problems can be grouped into three categories: (1) problems of alignment with the organization's objectives, (2) process problems, and (3) technical problems. Another survey of participants at successive software maintenance conferences presents a list of 19 key maintenance problems, ranked by importance, as perceived by software maintenance engineers (see Figure 1.1). The majority of the problems reported by this survey can be categorized in the maintenance process and management categories (items 3, 7, 8, 9, 10, 13, 14, 15, 16, 18, and 19 of Figure 1.1).

A second group of problems have been identified as originating from the software development process itself (items 4, 6, 11, and 12 of Figure 1.1). When the development quality is poor, the software is still transferred to the maintenance organization with a backlog of changes and problems that should have been addressed by the developers and which are hard to handle with a small number of individuals. Other maintenance problems stemming from the same source are:

- Poor traceability to the processes and products that created the software
- Changes rarely documented
- Difficulty of change management and monitoring
- Ripple effects of software changes

Rank	Maintenance problem
1	Managing changing priorities (M)
2	Inadequate testing techniques (T)
3	Difficulty in measuring performance (M)
4	Absent or incomplete software documentation (M)
5	Adapting to rapid changes in user organisations (M)
6	A large backlog of requests for change (M)
7	Difficulty in measuring/demonstrating the maintenance team's contribution (M)
8	Low morale due to lack of recognition and respect for maintenance engineer (M)
9	Not many professionals in the domain, especially experienced ones (M)
10	Little methodology, few standards, procedures and tools specific to maintenance (T)
11	Source code in existing software complex and unstructured (T)
12	Integration, overlap and incompatibility of existing systems (T)
13	Little training available to maintenance engineers (M)
14	No strategic plans for maintenance (M)
15	Difficulty in understanding and meeting user expectations (M)
16	Lack of understanding and support from IS/IT managers (M)
17	Maintenance software runs on obsolete systems and technologies (T)
18	Little will or support for reengineering existing software (M)
19	Loss of expertise when a maintenance engineer leaves the team or company (M)
	(M): Management Problem (T): Technical Problem

Figure 1.1 Problems as ranked by software maintainers [Dekleva 1992].

As software development processes progressively improve, there is renewed interest in improving the software predelivery and transition process. This process is requested by maintainers to protect them from taking over unfinished or low-quality software. The predelivery and transition activities address the maintenance-related issues identified during the software development process in order to ensure higher quality which, hopefully, will facilitate the final transition of the software to maintenance. By using such a process, the maintainer ensures that the developer will eliminate the defects before they arise or before time or budget run out to address them. The predelivery and transition process can be considered a quality control process. The lack of such a process can lead to costly maintenance contracts (or, conversely, to very profitable contracts for maintenance contractors).

Many software maintainers also consider that their professional status is perceived as being inferior to that of developers; for instance, they often get into a bind

and have no choice but to accept the newly developed software being forced on them, whatever its quality. The maintainers also report being poorly equipped, supported, and understood by management, while being responsible 24 hours/7 days for the proper functioning and management of software and, more importantly, for the support of all customer-related issues.

Frequent requests for changes by users (items 1 and 5 of Figure 1.1) are also identified as a source of problems; both the business world and the technology evolve rapidly, supported by quick changes in software and also in hardware. Finally, the difficulty in understanding and meeting user expectations is ranked fifteenth.

It is also reported in some organizations that the primary factors contributing to high software maintenance costs are: the number of programmers and their experience, the quality of technical documentation and user documentation, the tools that support maintenance engineers, the software structure and maintainability, and, finally, the contractual commitments that govern a product's maintenance.

Another problem is the impact of the work environment on programmer productivity. It has been reported that the physical environment (noise and interruptions) hinders their performance. Similarly, there is insufficient teaching of software maintenance skills in schools, and what is taught does not always reflect what software maintenance is in reality; this leads to a workforce with a lack of knowledge of maintenance-related processes and techniques.

Is there also a cultural factor that could explain why software maintenance is not sufficiently visible and not promoted? For instance, a study in Japan describes how managers' perceptions, based on the notion that software improvement increases customer satisfaction on a daily basis, seems to be a more important concept in a Japanese organization than it is in a European or American one [Bennett 2000]. Without good software maintenance, these Japanese organizations report that they would rapidly lose market share. When the quality culture is stronger, such as in Japan, maintenance activities benefit from more visibility and they achieve greater recognition: maintenance engineer morale is higher and maintenance management is more visible, both to managers and customers [Azuma 1994].

In summary:

- Nineteen problems have been identified and classified by maintainers, and most of them are managerial.
- Current practices in software-maintenance reporting do not highlight the value-added activities of software maintenance.
- Maintenance specialists' costs are significant and growing.
- There is still a poor understanding of the nature and contribution of software maintenance.
- Software complexity increases and quality decreases with multiple maintenance if no specific action is undertaken to address this issue.
- Low quality development has a direct impact on maintenance costs.

- The local organizational culture and its perception of the contribution of maintenance has an impact on its visibility and on maintenance staff morale.
- There is a lack of automated testing and diagnostic tools for successful change implementation.
- There is a lack of software maintenance training at the undergraduate level.

1.3 SOFTWARE MAINTENANCE BODY OF KNOWLEDGE

Every profession has a body of knowledge made up of generally accepted principles. In order to obtain more specific knowledge about a profession, one must either: (a) have completed a recognized education curriculum, or (b) have experience in the domain. For most software maintainers, software maintenance knowledge and expertise is acquired in a hands-on fashion in various organizations. Up to now, very few schools have offered extensive training on software maintenance. Even today, there is still very little in the way of a software maintenance education and training curriculum. Software maintainers and, in particular, software maintenance managers have had access to relatively few books, research papers, or teaching materials, with the exception of the few recent books of Jarzabek [2007]; Khan and Zhang [2005]; Grubb and Takang [2003]; Seacord, Plakosh, and Lewis [2003]; and Polo, Piattini, and Ruiz [2002]. Most other books on software maintenance were published 15 to 25 years ago, whereas generic books on software engineering refer to maintenance only as one life-cycle stage, their main focus being on software development.

The SWEBOK (Software Engineering Body of Knowledge [Abran et al. 2005]) and its Guide constitute the first international consensus developed on the fundamental knowledge required for all software engineers. The 2005 version of the SWEBOK itself can be obtained at http://www.swebok.org. In the SWEBOK, there is a knowledge area dedicated to the topic of software maintenance. This confirms that "Maintenance is a specific field in software engineering, and it is therefore necessary to study processes and methodologies which take the specific characteristics of maintenance into consideration" [Basili 1996]. The SWEBOK presents the taxonomy of software maintenance knowledge that is accepted within this specific domain knowledge of software engineering.

The first topic of software maintenance knowledge (see Figure 1.2) is labeled "fundamentals" and introduces the concepts and terminology that form the underlying basis for understanding the role and scope of software maintenance. The subtopics provide definitions and discuss the need for maintenance.

Categories of maintenance work are critical to understanding the underlying meaning of the many types of software maintenance. The second subtopic refers to the "key issues in software maintenance" that must be dealt with to ensure the effective maintenance of software. Software maintenance presents unique technical and management challenges to software engineers (e.g., the challenges inherent in trying to find a fault in a system containing one million lines of code that the maintainer did not develop or does not fully understand). Other examples of software maintenance challenges include the planning of a future release and its develop-

ment in parallel to emergency changes. In the SWEBOK, the maintenance issues are categorized into one of four categories:

1. Technical issues
2. Management issues
3. Maintenance cost and maintenance cost estimation issues
4. Software maintenance measurement issues

The third topic of the SWEBOK refers to the software maintenance process, including maintenance life-cycle models and activities, differentiating maintenance from development, and showing its relationship to other software engineering activities. It is labeled "Maintenance Process" and provides current references and standards used to implement this process.

The fourth topic refers to some of the generally accepted techniques for software maintenance: program comprehension, reengineering, and reverse engineering.

In the following sections, the definition and characteristics of software maintenance are presented.

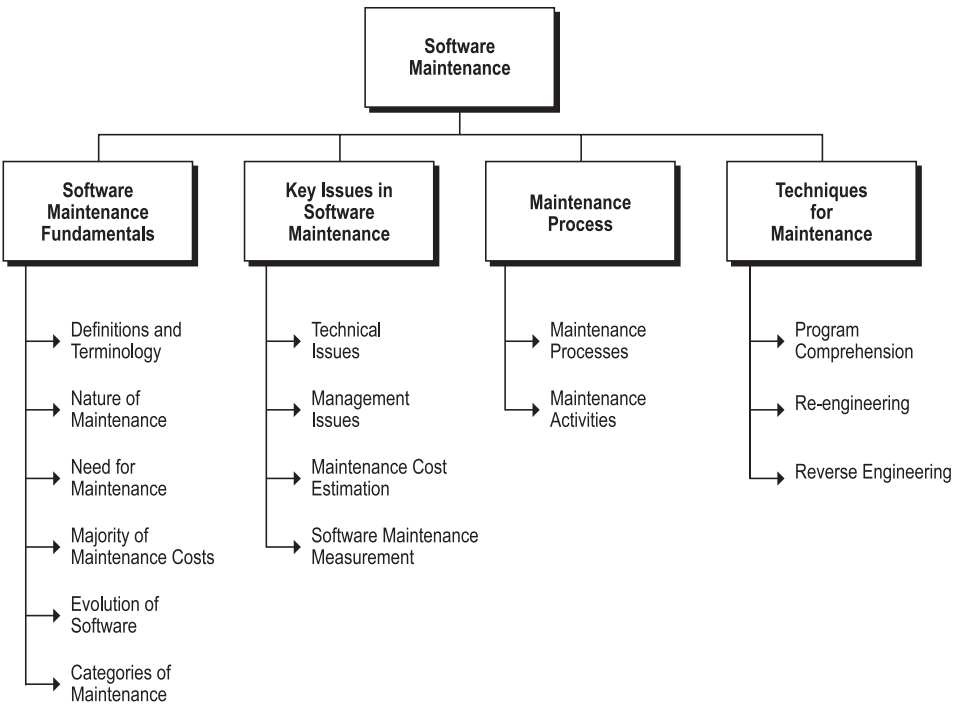

Figure 1.2 Software maintenance in the SWEBOK Guide [Abran 2005].

1.4 SOFTWARE MAINTENANCE DEFINITION

The software life cycle can be divided into two distinct parts:

1. The initial development of software
2. The maintenance and use of software

Lehman [1980] states that "being unavoidable, changes force software applications to evolve, or else they progressively become less useful and obsolete." Maintenance is, therefore, considered inevitable for application software that is being used daily by employees everywhere in a changing organization.

Some basic concepts and definitions of software maintenance are now introduced. Software maintenance is often focused on application software in our organizations. For example, application software contains business rules translated into functionality that is used and developed by the business to conduct its daily operations. It is quite different from other types of software that are not under the direct control of the organization or do not contain any business rules (e.g., operating systems and database management systems). We are not saying that these types of software do not require maintenance but it is not the typical business organization that maintains them. Figure 1.3 presents an overview of the often quoted definitions of software maintenance (organized chronologically).

Definition - Interpretation	Author	Year
"Changes that are done to a software after its delivery to the user"	Martin & McLure [Martin 1983]	1983
"The totality of the activities required in order to keep the software in operational state following its delivery"	FIPS [FIPS 1984]	1984
"Maintenance covers the software life-cycle starting from its implementation until its retirement"	von Mayrhauser [von Mayrhauser 1990]	1990
"…modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity."	ISO12207 [ISO 1995]	1995
"…the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."	IEEE1219 [IEEE 1998a]	1998
"…the totality of activities required to support, at the lowest cost, the software. Some activities start during its initial development but most activities are those following its delivery."	SWEBOK [Abran 2005]	2005

Figure 1.3 Generally accepted definitions of software maintenance.

1.5 DIFFERENCES BETWEEN OPERATIONS, DEVELOPMENT, AND MAINTENANCE

In an organizational context, the day-to-day maintenance and development of application software developed internally are typically the responsibility of an operational support unit within the software organization. For software acquired externally, the software maintenance is not done by the organization using the software. There is sometimes confusion about who conducts the maintenance and where maintenance activities start and end.

Computer operations activities are considered as distinct from software maintenance activities; for example, the ISO 14764 international software maintenance standard specifies that operational activities (backups, recovery, and systems administration) are handled by the computer operations team, and, therefore, fall outside the standard scope of software maintenance. This scope is similar to what is often mentioned in the literature and, in practice, managers or customers clearly distinguish computer operations from software maintenance activities.

There are still, however, important interfaces between the two, to ensure that the infrastructure supporting the software is functional and efficient (e.g., change management, support calls concerning system failure, recovery of the environment and data in case of disaster, data recovery and work restart, processing time investigations, automated scheduling, and management of disk space and tape libraries) [ITIL 2007b].

Differences between software maintenance and software development activities are sometimes more difficult to distinguish clearly. Software development possesses an interface with software maintenance, which in some organizations is sometimes imprecise and ill-defined. At times, the differences are blurred when the software developer is also involved as well with the maintenance of the software. This blurring is caused by certain maintenance activities that are similar to those found in development (analysis, design, coding, configuration management, testing, reviewing, and technical documentation). Although some activities are similar in theory, they still must be tailored to a maintenance context; for example, small maintenance tasks are being most often performed by one or two programmers with very short-term deadlines and not by a project team with a much longer planning horizon. Significant feature enhancements are rarely done this way as they require a maintenance project approach.

Some authors have studied the differences and similarities between software development and maintenance activities. The main findings are that, in development, the organizational structure mainly takes the form of a project, with a beginning and an end using a team structure to deliver results within an approved budget and time frame. The typical software development project is created for a fixed, temporary period of time, and does not persist past software delivery. The project team develops a plan for resources, costs/benefits, fixed deliverable objectives, and aims for the planned project completion date.

The maintenance organization unit, however, is structured to meet quite different challenges, such as randomly occurring daily events and requests from users, while providing continued service on the software for which it is responsible. Some of the

unique characteristics of software maintenance, as compared to development activities, are [Abran 1993a]:

- Maintenance requests (MRs) come in on an irregular basis, and cannot be accounted for individually in the annual budget planning process.
- MRs are reviewed and prioritized, often at the operational level. Most do not require senior management involvement.
- The maintenance workload is not managed using project management techniques but, rather, queue management techniques.
- The size and complexity of each small maintenance request are such that it can usually be handled by one or two maintenance resources.;
- The maintenance workload is user-services oriented and application-responsibility oriented.
- Priorities can be shifted around at any time, and requests for corrections of application software errors can take priority over other work in progress.

In software maintenance, operational events that cause a software failure are reported in problem reports (PRs). However important the estimated effort may be, PRs are handled by the maintenance team immediately. Whatever the size or effort required to fix a failure identified in a PR, it will be attended to immediately by the maintenance staff. This maintenance service has priority over all other services.

In addition, a user can request a modification to the software by issuing a modification request (MR). The request will go through a number of assessment steps before being accepted. The maintainer will first conduct an assessment to estimate the effort needed to modify the existing software. Dorfman and Thayer's study [1997] indicates that MRs and PRs go through an investigation and impact analysis process that is unique to software maintenance. Impact analysis describes how to conduct a complete analysis of a change in existing software [Koskinen 2005b, Visaggio 2000, Arnold and Bohner 1996]. In the case of the modification, if the estimated effort were too great, the request would be rerouted to a development team to be treated as a small software development project. There is, therefore, in software maintenance, a unique process of acceptance/refusal and classification of work for MRs (see Figure 1.4).

This process takes into consideration as an input the estimated size of the modification [Maya et al. 1996]. April [April 2001] presents the process used in a Cable & Wireless member company, in which the maximum acceptable effort for an adaptive request is five days. This five-day limit is also recognized by the United Kingdom Software Metrics Association (UKSMA):

> The distinction between maintenance activity of minor enhancements and development activity of major enhancement is observed in practice to vary between organizations. The authors are aware that in some organizations an activity as large as to require 80 to 150 workdays is regarded as maintenance, while in others the limit is five days. Initially it is proposed that the ISBSG and UKSMA will adopt the convention that work requiring five days or less will be regarded as maintenance activity. [ISBSG 2007]

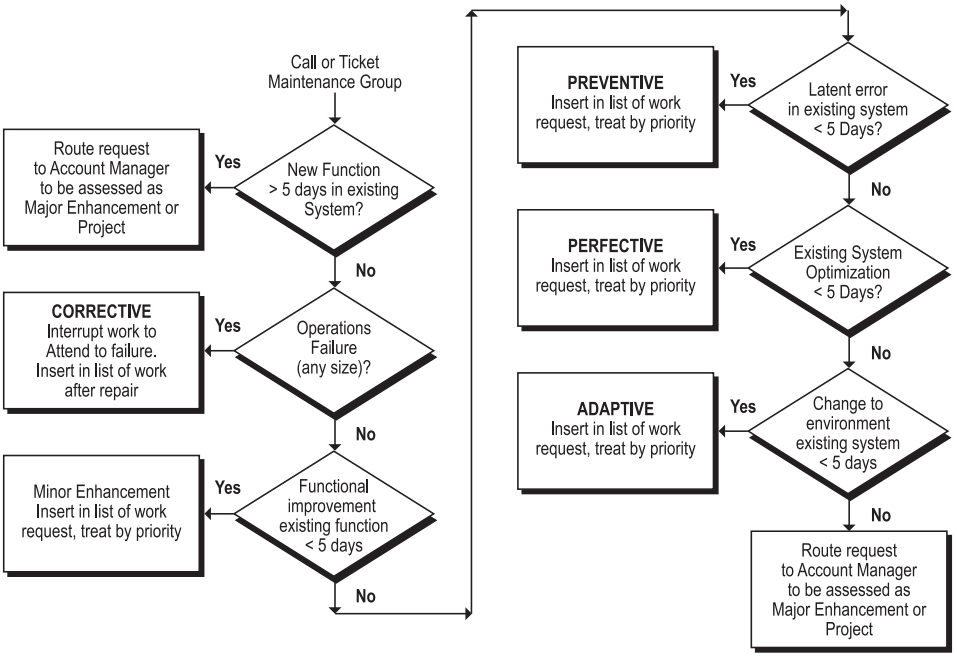

Figure 1.4 Example of the acceptance/refusal process of maintenance work [April 2001].

This very small limit is not likely to exist for maintenance on complex military, nuclear, aerospace, and aircraft software maintenance as it may occur over multiple months or years. The authors acknowledge that they have not taken those industries into consideration when selecting the five-day limit.

Such a threshold is very important in IS/IT organizations, as it dictates when software development starts and when software maintenance ends. For the maturity model proposed later on in this book, no limit in days should be specified. What is essential is that the maintenance work be identified, whatever its estimation effort, and be performed by individuals and not project teams.

Other activities are considered as specific to software maintenance, such as change requests supported by a help desk call center and its support software, activities evaluating the impact of changes, and the specialization in testing and regression checks.

Additional differences in software maintenance have been identified, such as (see Figure 1.5):

- Problem management
- Acceptance of software
- Management of the development's transition to the maintenance team

Some maintenance Activities	Software management (maintenance)	Software development (creation)
Management of problems (problem resolution interfacing with a help desk)	P	A
Acceptance of the software	P	A
Managing transition from development to maintenance	P	A
Establisment of service level agreements (SLAs)	P	A
Planning of maintenance activities (versions, SLA, impact analysis)	P	A
Managing events and service requests	P	A
Supporting daily operations	P	A
Rejuvenating sofware	P	A

Figure 1.5 Software maintenance activities (P = present, A = absent).

- Role of the users, the operations team, and the maintenance employees
- Management of maintenance
- Software management (improvements and performance)

Software maintenance can also be described as a service with the following characteristics [Bouman 1999]:

- Emphasis on direct sale to the user
- Frequent and direct contact with the user
- Service supplied immediately, rather than a few months down the road
- Short service time
- The product not always a physical good
- The product not always fit for storage or transport
- Services more specialized and more crafted than physical goods

In summary, software maintenance comprises processes and activities that are not handled by software development groups. Software maintenance also uses processes and activities of software development, especially while implementing a modification to existing application software [ISO 1995, s5.5.3]. It can also be concluded that maintenance is a specific domain of software engineering accepted in the SWEBOK Guide and ISO international standards. It is therefore necessary to examine some of the processes and methodologies that take the specific characteristics of software maintenance into consideration.

1.6 WHICH ORGANIZATIONAL UNIT IS RESPONSIBLE FOR SOFTWARE MAINTENANCE?

A software maintainer is defined, in ISO 12207, as an organization that performs maintenance activities. In the industry, software organizations currently favor two organizational structures with regard to the location of the software maintenance function. The first model favors maintenance of the software by its developer. In another model, a software maintenance organization, independent of the developer, takes care of the organization's software maintenance needs. Advantages and disadvantages of each type of maintenance model are described in Abran et al. [2005] and Pigoski [1997].

There are a number of disadvantages to letting the development team maintain the software after it has been put into production:

1. Developers do not like performing maintenance and are more likely to leave for more interesting work.
2. New hires in the development team will be both surprised and dissatisfied to discover that they also need to maintain existing software.
3. Developers are often reassigned to other development projects and prefer this kind of work.
4. When the individuals who developed the software leave, the other employees will probably not be qualified to maintain it.

In management meetings at a Cable & Wireless member company during 2003, it was often mentioned that development organizations that also conduct the maintenance lack transparency and that their products are perceived as being of lower quality, lacking proper documentation, and leaving little flexibility for knowledge transfer between programmers. Pigoski [1994] as well as Martin and McClure [1983] and Swanson and Beath [1989] also concluded that it is in the best interests of large organizations to create an independent software maintenance organization, as they will get better results in terms of quality and independence (e.g., documentation, formalities of transition, specialization of maintenance programmers, management of requests for changes, and overall employee satisfaction). It is not clear, however, at what organization size it becomes preferable to switch to the organizational model in which maintenance is independent of development.

1.7 SOFTWARE MAINTENANCE STANDARDS

Software maintenance is considered as one of the five primary processes in the software life cycle processes of the ISO 12207 international standard. In this international standard, the software engineering processes are divided into three main groups: primary, supporting, and organizational. In ISO 12207, the primary processes include acquisition, development, maintenance, and software operation

activities. The support processes include such activities as documentation writing, configuration management, quality assurance, verification and validation, review, audit, and software problem resolution. Finally, the organizational processes are typically offered to the whole organization. These organizational processes include general training, infrastructure, process improvement, and management activities.

In ISO 12207, the software maintenance process includes six major subprocesses (see Box 5.5 in Figure 1.6):

- Process implementation
- Problem and modification analysis
- Modification implementation
- Maintenance review/acceptance
- Migration
- Software retirement

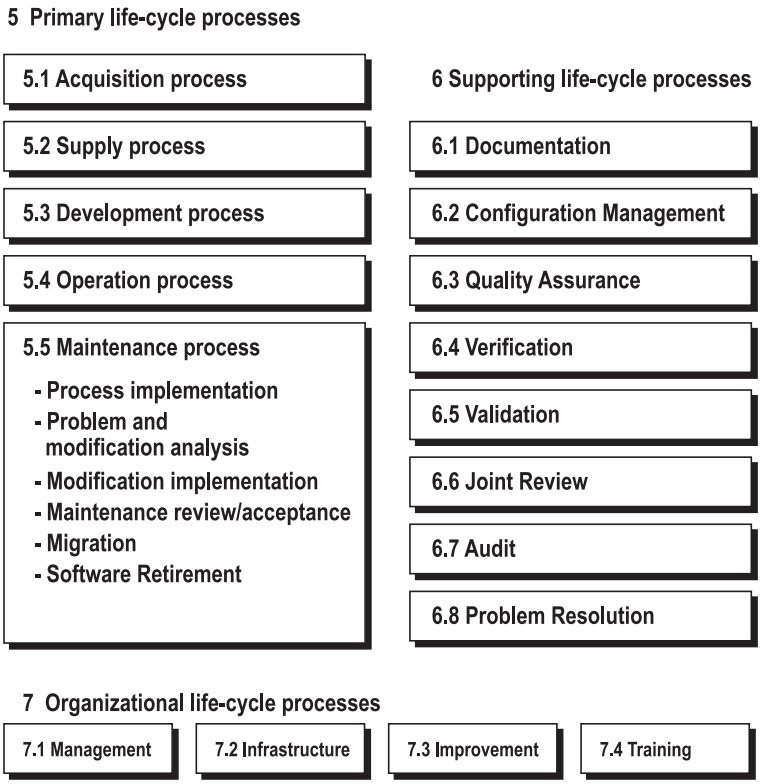

Figure 1.6 Software maintenance as a primary process of ISO/IEC 12207 [ISO 1995].

The maintenance process may also call on other primary, supporting, or organizational processes when needed. This standard also clarifies which of the activities, also shared by developers, should be adapted when used by maintainers (i.e., documentation, configuration management, quality assurance, verification, validation, reviews, audits, problem resolution, process improvement, infrastructure management, and training).

The ISO 12207 standard also suggests that the maintainer can adapt other software engineering subprocesses that come from:

- A development process
- A management process
- An infrastructure process
- An improvement process
- A training process
- A supply process

ISO 12207 does not, however, provide guidance on how to do this; more specific guidance to software maintenance engineers is provided in the ISO 14764 international standard.

What clarifications are provided by ISO 14764 in regard to the relationship between maintenance and development (ISO 12207's *primary process*)? The international software maintenance standard ISO 14764 specifies that when development standards are being used by maintainers they must be adapted to meet the specific requirements of maintenance [ISO 2006, s8.3]. The more specific references in ISO 14764 and ISO 12207 are as follows: in ISO 12207, it is notably specified that the analysis used to determine which part of the software and documentation will be modified (s5.5.3.1) must use development processes, and section 5.3 describes the development process associated with the documentation of the test, test results, and test review activities that must also be called upon (s5.5.3.2).

The ISO software engineering standards (i.e., ISO 14764) are oriented toward "classic" activities and do not address certain aspects of the processes that support software maintenance as experienced in industry. Additionally, a significant number of standards are used by software maintainers. The most relevant IEEE and ISO standards, as well as their interdependencies, are presented in Figure 1.7. James Moore explains in his book, *The Road Map to Software Engineering: A Standards-Based Guide* [Moore 2005], the layered diagram he developed to show the relationships between different standards "from the general to the more specific." We have used the same technique here to come up with a similar representation from the software maintainers' perspective. James has kindly agreed to review it for us.

In this representation, we see that maintenance could utilize other software engineering processes quoted in at least 40 other standards to conduct its daily activities. So, how can we explain that the majority of maintainers do not currently use software maintenance element standards and other referenced standard processes? One tentative explanation could be that it is partly because the maintainers "inherit" not

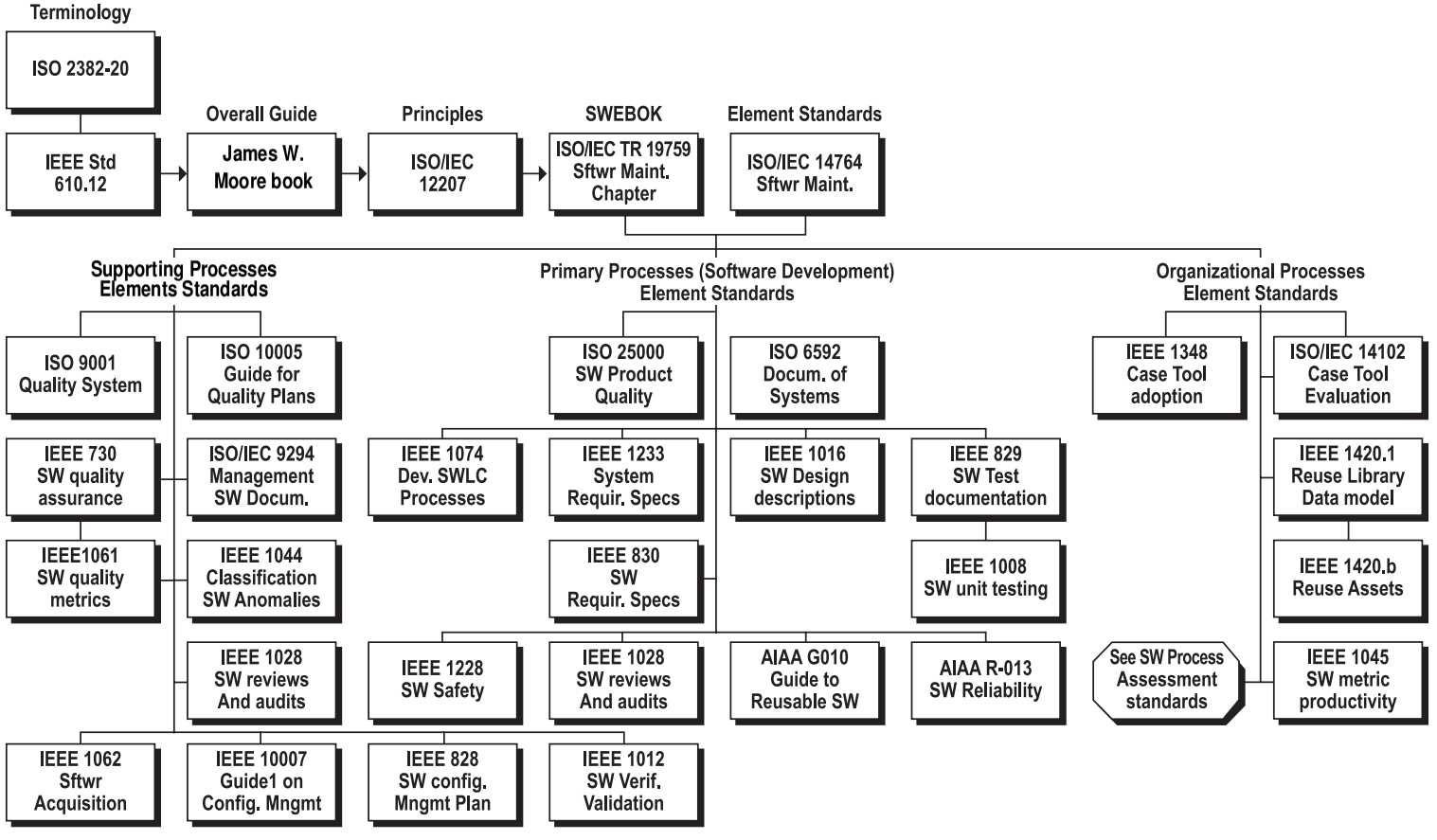

Figure 1.7 Software standards that could be of use to software maintainers.

only the legacy code but the legacy processes to go with it. Hence, the maintainers tend to follow the "standards" set forth in the legacy code. Another tentative explanation is that most maintenance organizations demonstrate all the characteristics of a low-maturity-level organization and are mostly reactive rather than proactive in the management of software maintenance.

On the other hand, a few organizations at the highest CMMi maturity level would use standards extensively, such as in the space shuttle organization, in which the team has been together for years, and technology does not change much.

With such a plethora of candidate standards, Schmidt, in his book, *Implementing the IEEE Software Engineering Standards* [Schmidt 2000], argues that it is not realistic to consider using all available standards right away when implementing formal processes in a software organization. Schmidt argues that organizations will adopt a progressive use of software engineering standards, suggesting that there may be a direct relationship between an organization's maturity and its use of standards. He, therefore, proposes a pyramid representation that describes the progressive use of standards based on four factors: (1) the application's criticality level, (2) team size, (3) the maturity of existing processes, and, finally, (4) external process requirements (see Figure 1.8). We have used this representation to show what type of pyramid we may get from a software maintainer's point of view.

In this model, the processes become more and more rigorous as the criticality level of the software increases. Similarly, as the size of the teams increases, more and more standards will be needed in order to establish proper support for increasingly important processes and internal communications. An organization's maturity

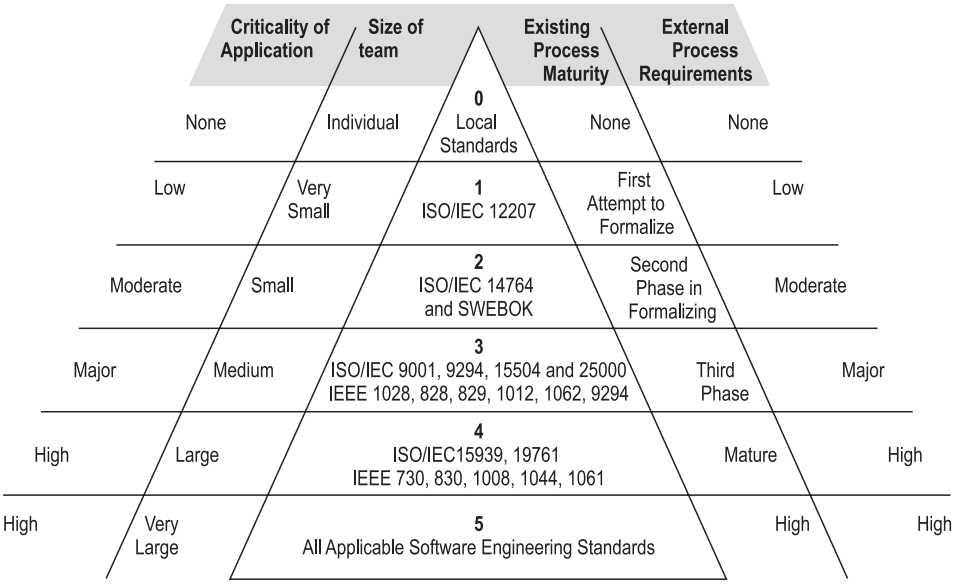

Figure 1.8 Applicable standards pyramid—a simple maturity model.

will also have an impact on the number of standards that can realistically be integrated into existing employee operational processes without being rejected/resisted. Finally, external requirements can enforce the use of certain standards when the products do not conform to certain industry-specific regulations (e.g., nuclear, medical, aeronautical, or defense).

Level 0 describes the adoption of very high-level processes and is characterized by the absence of formal standards. It is only possible to be on this level if there are no external requirements and the software has low criticality. At this level, it is possible to describe the maintenance work using mostly the ISO 12207 standard, which allows for the presentation of a general model of software activities. This ISO 12207 standard does not delve into the details of how activities are accomplished in a specific organization. The other five levels of the standards pyramid are:

Level 1 standards are used for the study of requirements and the documentation of tests.

Level 2 progressively adds formality by introducing work plans, documentation of concepts, and the use of a systematic approach for reviews.

Level 3 goes even further by proposing the use of standards for quality assurance, verification and validation, configuration management, and security issues.

Level 4 proposes additional standards for the life cycle, configuration management, unit testing, and two important standards for measuring software quality.

Level 5 proposes the use of all currently published IEEE standards.

Schmidt's model could apply to software maintenance (because software maintenance directly refers to activities for the development process in ISO 12207. His model is a theoretical proposition, in the sense that it is supported by neither experiments nor implementations; it still provides an expert opinion in support of maturity model concepts being potentially supported by the many software engineering standards.

Some have argued that that maintenance processes reflected in current international standards approximately match level 2 practices of the SEI model. It is also suggested that a direct relationship exists between the maturity of an organization and its progressive use of standards concepts.

In summary, a large number of software standards can be referenced by maintainers. One international standard is central to software maintenance (ISO 14764). The software maintenance domain refers to software development standards when needed. Maintainers must, therefore, use software development standards while making sure to adapt them to the specific maintenance context [ISO 2006 s8.3.2.1 and s8.3.2.2]. Incidentally, by nature, standards do not include topics on emerging technologies or those that are not perceived as generally accepted. The list of unique software maintenance activities are candidates for inclusion in a future version of the ISO 14764 standard as the members of the ISO/JTC1/SC7 committee accept them as generally accepted.

1.8 SOFTWARE MAINTENANCE PROCESS AND ACTIVITIES

Although software maintenance international standards have been available for more than 20 years, many organizations still do not have a well-defined process for software maintenance activities. Van Bon [2000] has also reported that there is little in the way of software maintenance process management and that the domain continues to be neglected.

In the early days of the software industry, there was no difference between software development and software maintenance. In the beginning, legacy code was so small you could rewrite it every time there was a change needed! And programmers did it for fun. Only in the 1970s did life-cycle models specific to the software maintenance process start to appear. In general, these first models were broken down into three steps: (1) understanding, (2) modifying, and (3) validating changes to software. Although many development methodologies still do not represent the software maintenance step, others have included software maintenance as their last step. From the software-life-cycle process point of view, the IEEE 1074 standard considers maintenance to constitute the seventh of eight phases of the development project cycle, at the end of the implementation phase of a new piece of software. However, some have suggested that "the traditional view of the software development cycle has done a great disservice to the maintenance domain by representing it as one step at the end of the cycle" [Schneidewind 1987].

The 1980s brought models oriented toward the maintenance process [Bennett 2000, Fugetta 1996]. Maintenance was no longer seen as the last step in development. These models present maintenance-specific activities and introduce precedents for each of them. Some consulting organizations have also defined their own maintenance standards, which comprises variants of maintenance-specific activities.

The 1990s saw the emergence of the international standards (ISO 12207 and ISO 14764]) that are being used today. In its current version, the ISO 12207 standard represents maintenance processes that overlap many other phases of the software-life-cycle processes, in the sense that its realization may require going through, partially or totally, most of the software-life-cycle processes. To better situate and present the boundary of the software maintenance processes in ISO 14764, a general diagram is presented in this standard (Figure 1.9).

The ISO 14764 process model includes the six key processes of software maintenance. Individual organizations and suppliers can, of course, adapt this process model, which must generally meet the international standard.

Let us look at each process in more detail. The implementation process is external to the more operational processes (shown inside the circle) of software maintenance. It contains software preparation and transition activities, such as the design and documentation of the maintenance plan, the preparation for handling problems identified during development, and the follow-up on product configuration management.

There are three operational processes within this process model, as shown inside the circle in Figure 1.9:

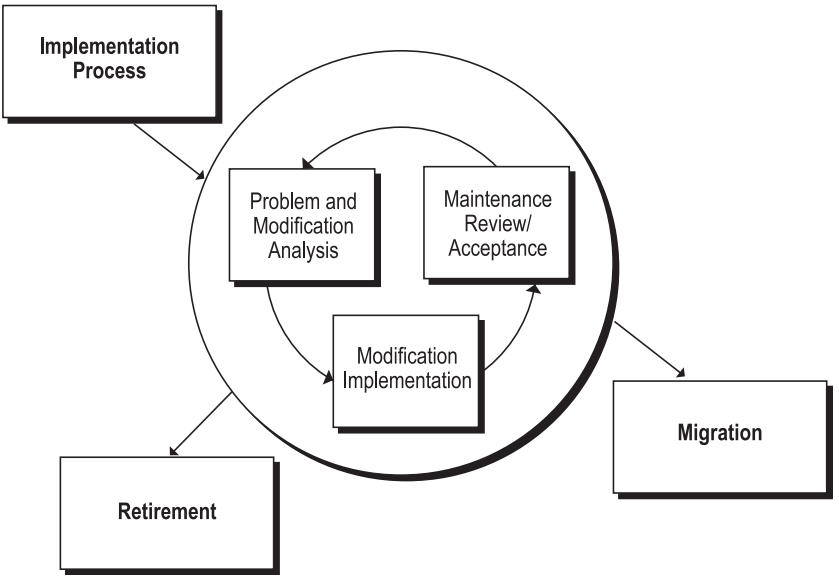

Figure 1.9 Software maintenance key processes [ISO 2006].

1. The problem and modification analysis process, which is executed once the application software has become the responsibility of the maintenance group. The maintenance programmer must analyze each request, confirm it (by reproducing the situation), check its validity, investigate it, propose a solution, document the request and the solution proposal, and, finally, obtain all the required authorizations to apply the modifications.
2. The modification implementation process considers the implementation of the modification itself. This process is more of a development-specific process (with a direct reference to the development process activities in ISO 12207, section 5.3).
3. The maintenance review/acceptance process for the acceptance of the modification. This consists of a verification and approval with the individual who submitted the initial request, and ensures that the solution provided meets the initial requirements.

Migration processes (platform migration, for example) are less frequently used, and are not part of the daily maintenance tasks. If the software must be ported to another platform without any change in functionality, these processes will be used and a maintenance project team is likely to be assigned to the specific project task.

Finally, the last maintenance process, an event which does not occur on a daily basis, is the retirement of a piece of software. Both the migration and retirement processes are typically handled as projects, not as small maintenance work, since the effort usually requires that thresholds be set for work classified as small mainte-

nance requests. From this perspective, some activities of software maintenance are quite different from daily software support.

As discussed elsewhere in this chapter, a considerable number of the maintenance activities mentioned in the literature have not yet been standardized and, therefore, are not represented in the process model in Figure 1.9 but are observed in industry.

1.9 SOFTWARE MAINTENANCE CATEGORIES

In 1976, E. B. Swanson [Swanson 1976] was one of the first to come up with a three-category classification of maintenance activities, which he labeled corrective, adaptive, and perfective. This was followed by a survey that classified maintenance work into four categories: corrective, adaptive, perfective and preventive. Today, ISO 14764 still uses these maintenance work categories (see Figure 1.10).

More detailed categories of maintenance work have since been proposed recently [Chapin 2001]. Other approaches, like ontology, have also emerged to describe further the software maintenance processes and activities. An ontology (see Figure 1.11) represents a view of a domain in which the concepts are defined in a precise and sometimes formal way. A number of publications present concepts of ontology applied to software maintenance [Kitchenham 1999, Derider 2002, Vizcaíno 2003, Dias et al. 2003, and Ruiz 2004].

1.10 MAINTENANCE MEASUREMENT

Measurement is fundamental to Deming's view of process control: "A process is stable if its future performance is predictable within statistically established limits" [Deming, 1986]. By measuring, we can learn more about the behavior of resources, the process, and the software product, and their relationships of cause and effect (see Figure 1.12).

1.10.1 Maintenance Process Measurement

A process is a sequence of steps performed for a given purpose. It is generally accepted that the quality of software is largely determined by the quality of the development process used to design it. The maintenance manager's objective, then, is to

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptative

Figure 1.10 ISO 14764 software maintenance categories.

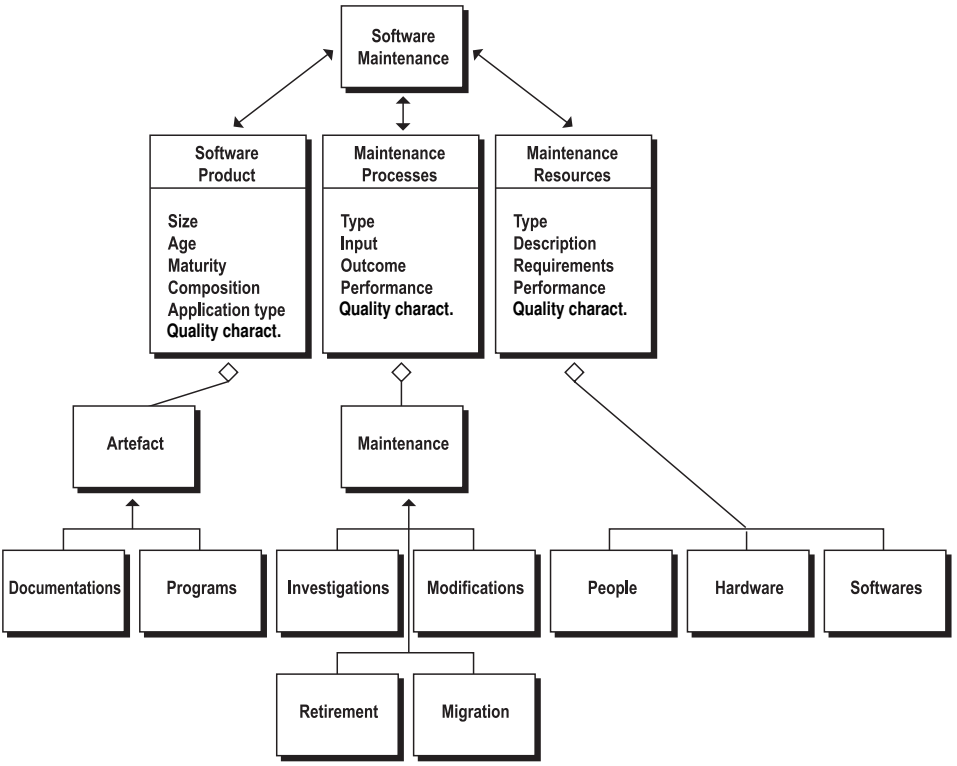

Figure 1.11 Ontology of software maintenance [Kitchenham 1999].

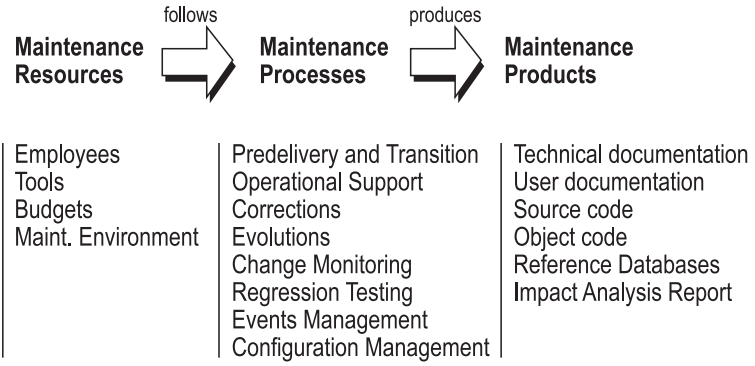

Figure 1.12 Software measurement perspectives.

help bring such a process under control, and measurement has an important role to play in helping to meet this objective. Because he or she has little control over the development of the software, the maintainer must identify the earliest point at which it is possible to influence the maintainability characteristics of the new software under construction. Initiating measurement during predelivery and transition could be the strategy to use to assess the quality of the product and its readiness to be accepted in maintenance. To achieve this goal, a decision can be made to implement a software maintenance measurement program and link it to the software development measurements. For example, if the maintainer can set some maintainability objectives early on in the development of new software, its quality could be measured both during and after development. For all of us, this would be an ideal situation. At the highest maturity levels, a maintainer would be in that situation. We can only hope that the rising maintenance costs will motivate organizations to improve considerably their maintenance processes.

Many factors must be taken into account before measuring software maintenance processes. One strategy is to identify the key activities of a given process and their improvement goals. These key activities, in turn, have many characteristics that can be measured. Desharnais and coworkers [Desharnais et al. 1997] show an example of the software maintenance characteristics they chose for a pension plan agency in Canada (see Figure 1.13). They recommended that measures be well de-

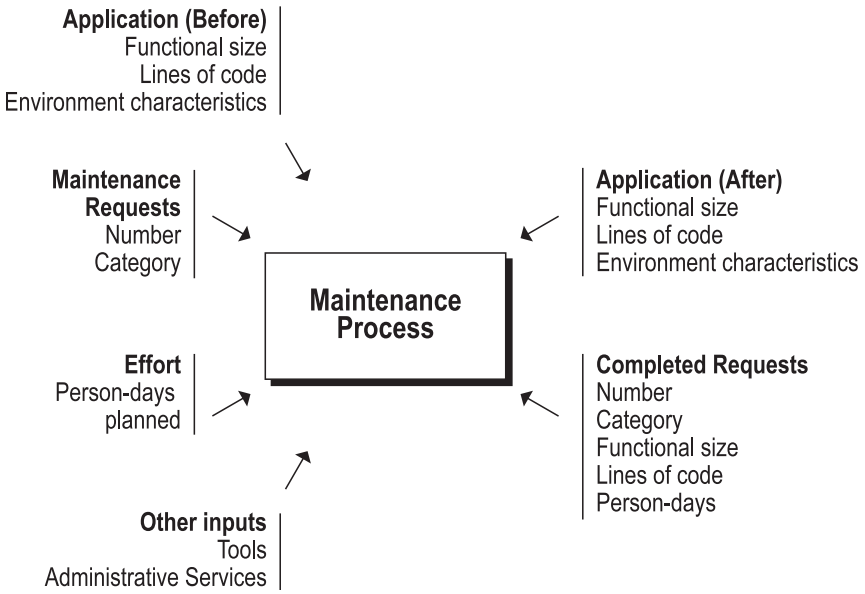

Figure 1.13 Example of selected characteristics of a maintenance process [Desharnais et al. 1997].

fined and verified to ensure that they properly characterizse the product, the services, and the maintenance processes. But before measures are well defined, it is essential that processes and products be defined first.

Maintenance measurement prerequisites have been presented in April and Al-Shurougi [2000], together with requirements in terms of (1) quality goals, (2) definitions of work categories, (3) implementation of request management process/software, (4) classification of maintenance effort in an activity account data chart (billable/unbillable), (5) implementation of activity management (time sheet) software and data verification, and (6) the measurement of the size of requests for change.

Many organizations often rely too much on qualitative information from internal reviews to evaluate software quality when they should be using quantitative methods instead. "Low maturity organizations often collect measures just to show data. It is rarely used in practice" [Ebert 2004 p. 174]. The SEI CCMi describes process measurement activities as appearing at level 2 maturity and refers to the necessity of a basis of maintenance process before measurement can be initiated.

The measures proposed by the SEI were suggested from a development perspective and do not capture the peculiarities of software maintenance. Other authors [McGarry 1995, Desharnais et al. 1997] also confirm this view and specify that a software maintenance measurement program must be planned separately from that of the developers. The measurement requirements being different, software maintenance measures are more focused on problem resolution and on the management of change requests.

High maturity organizations established maintenance measurement programs as early as 1987. Such a measurement program at Hewlett Packard is documented in Grady and Caswell [1987, p. 247], illustrating that measurements must be based on well-documented software processes. Grady and Caswell highlight that a process that is not documented, is difficult if not impossible to improve using measures. Hewlett Packard implemented a company-wide measurement program, including a description of the collection of data specific to software maintenance. Grady and Caswell identify the following concerns that are specific to a software maintenance measurement program:

- How should we plan for staffing maintenance of a software product family?
- When is a product stable?
- Is mean time between failures (MTBF) for a software product really a meaningful measure of software quality?
- In which development phase should tool and training investments be made?
- How does documentation improve supportability?
- How many defects can be expected in a project of a given size?
- What relationships exist between defects found prior to release and those found after release?
- What, if any, is the relationship between the size of a product and the average time to fix a defect?

- What is the average time to fix a defect?
- How much testing is necessary to ensure that a fix is correct?
- What percentage of defects is introduced during maintenance? During enhancements?

"Estimation is still one area in software measurement where many struggle mightily—right from the start" [Ebert 2004]. Software maintainers are no different in this respect. The estimation of software maintenance resources (effort, number of employees, and budget) is typically conducted by software maintenance engineers using one of two approaches [Abran and Maya 1995, Sellami 2001]. The most common estimation approach in software maintenance is based on experience, that is, individual recollection as a basis to determine maintenance requirements, extrapolate their impacts, and estimate expected costs. Another approach consists of using maintenance-specific parametric models (such as the COCOMO maintenance model [Boehm 1981] or similar proposals [Hayes et al. 2004, Chan 1996]). However, there are few maintenance organizations that report using these tools in their daily work.

Experience is often used as a basis for estimation in software engineering, as there is a lack of quality historical data. The following paragraphs illustrate some types of measurements that are needed in software maintenance and the need to compile historical data that could be used to build software maintenance estimation models.

A number of authors present specific measures that are useful to maintainers. In a survey on the measurement of maintenance, Dutta and coworkers [Dutta et al. 1998] report that 75% of individuals surveyed indicated that they documented the number of failures, analyzed their cause, and identified the origin of failures in production software. Another ratio of interest is the *average change time,* which is the average time it takes to implement a change in application software after its initial development; application software with a shorter average change time is considered to have better maintainability.

Hitachi reports using a maintainability cost measure (e.g., cost of changing software after its development) by using the ratio of the number of failures to the cost of the initial development project. By measuring multiple maintenance projects from the same organization, Tajima [1981] could then document measures to identify which application software has better maintainability than others.

Authors [Abran 1991, Abran 1993a, St-Pierre 1993, Desharnais et al. 1997] also report individual measures and software maintenance portfolio measurement grouped by software maintenance category. They identify trends within each of these categories and used these trends to explain the maturity profile of software.

Maintainers performing maintenance of the Space Shuttle software have lots of experience on the software they maintain, have worked with the same team for years, and have not upgraded the technology of their maintenance process for a long while. This is the kind of environment that operates at a high level of process maturity. NASA proposed 12 measures for corrective and adaptive software maintenance [Stark et al. 1994]: "application size, number of employees, number of re-

quests/status, number of perfective requests/status, use of computer resources, bug density, volatility, issue report open time, failure/repair ratio, reliability, design complexity, bug type distribution."

1.10.2 Software Product Measurement

The quality of a software product is of particular importance to maintenance teams. The ISO 9126 [ISO 2001] standard identifies six characteristics of a software product's quality, including maintainability (Figure 1.14). This standard is currently under review by Work Group Six of ISO/JTC1/SC7. The maintainability characteristic is further described by four subcharacteristics: analyzability, changeability, stability, and testability. "Maintainability is not restricted to code; it describes many software products, including specifications, design, and test plan documents. Thus we need maintainability measures for all of the products we hope to maintain" [Pfleeger 2001].

Two different perspectives of maintainability are often presented in the software engineering literature. From an external point of view, maintainability attempts to measure the effort required to diagnose, analyze, and apply a change to specific application software. From an internal product point of view, maintainability is related to the attributes of application software that influence the effort required to modify it. The internal measure of maintainability is not direct, meaning that there is no single measure for the application's software maintainability and that it is necessary to measure many subcharacteristics in order to draw conclusions about it.

The IEEE 1061 [IEEE 1998b] standard also provides examples of measures without prescribing any of them in particular. By adopting the point of view that reliability is an important characteristic of software quality, the IEEE 982.2 [IEEE

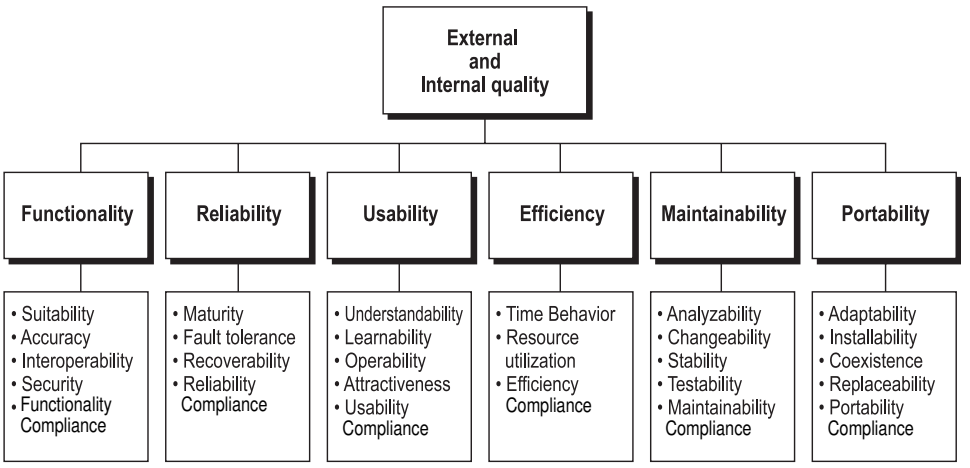

Figure 1.14 ISO 9126—Software product quality model [ISO 2001].

1988] guide proposes a dictionary of measures. This standard identifies, among other things, six distinct measures of source code complexity.

Internal measures often describe the structural complexity of software. Structural complexity measures are generally extracted from the source code using observations on the program/module/functions graph of software source code. Studying graphs allows for collecting information on the complexity of the application software. There is a large body of literature on the static evaluation of source code. These studies are based on work carried out in the 1970s by McCabe, Halstead, and Curtis [McCabe 1976, Halstead 1978, Curtis 1979]. Since then, a number of researchers have proposed a large number of measures more adapted to an object-oriented environment.

Maintainers, managers, and especially users are looking for a single-number approach that indicates the relative maintainability of a given software; they basically wish to use this number to help in the decision to accept/reject the transition of a new software product from development to maintenance. For instance, some have proposed a maintainability index, which is a composite measure incorporating a number of traditional source-code measures into a single number [Welker 2001]. In practice, however, the single-number approach is quite elusive and has been found to be of use at best as a very rough indicator. Much further research is still required in this area.

There are a number of software tools available in the industry to measure source code complexity. These include ready-to-use measures and also tools that allow the user to create new ones for specific requirements. However, interpreting these measures is very difficult since they are very specific and there are few mechanisms to summarize the data for decision making. Managers and customers often end up with technical measures that do not add much value to be communicated to the software user.

In summary:

- The measurement of software products is a specialized domain; the ISO 9126 measures were published in the early 2000s and maintenance organizations still have to put them into practice.
- Tools to measure quality of source code are available commercially.
- Interpreting source code measures is still difficult, since they are very specific and few mechanisms exist to summarize the data for decision-making purposes.
- Subjective measures still play an important role in assessing software maintainability.

1.11 SERVICE MEASUREMENT

Some authors claim that “software maintenance has more service-like aspects than software development” [Niessink 2005]. This seems to be the case for other information technology services as well, like computer operations. Service quality mea-

sures for software maintenance services have been proposed in the literature and divided into three categories:

1. Internal service-level agreement
2. Maintenance service contract
3. Outsourcing contract

1.11.1 Internal Service-Level Agreement

To reach agreement on service levels, a consensus must be developed between the customers and the maintenance organization about the related concepts, terms, and measures to be used. Service-level agreements (SLAs) are said to be internal when they are exercised entirely within a single organization. This type of agreement documents consensus about activities/results and targets of the many maintenance services. SLAs originally appeared in large computer-operations centers during the 1950s. They progressively extended their reach to all IS/IT service-oriented activities during the 1970s [ITIL 2007b]. However, many IS/IT organizations still do not have SLAs in place today [Karten 2007].

Some attempts to identify exemplary practices and to propose SLA maturity models have appeared recently. CobiT® [IT Governance Institute 2007] is a set of exemplary practices used by IS/IT auditors. CobiT® identifies and describes practices that must be implemented in order to meet the IS/IT auditor requirements for SLAs. It identifies the *definition and management of the service level* as an essential practice, with the measurement of quality of service as its main objective. CobiT describes five maturity levels for the agreement (nonexistent, ad hoc, repeatable, defined, managed/measured, and optimized). CobiT® also describes other software engineering processes that are directly related to software maintenance. Other proposals of SLA maturity models that have recently been put forward can be found in Niessink et al. [2005] and Kajko-Mattsson et al. [2004].

SLAs become an important element of customer satisfaction in a competitive environment. A few publications have directly addressed SLAs in a software maintenance context [Bouman 1999, CobiT 2007, Niessink 2000, April 2001, Niessink et al. 2005, Kajko-Mattsson et al. 2004].

Even though many publications have presented detailed elements of SLA, there is still little known about the key underlying principles at work in these types of agreements. Bouman [Bouman 1999] presents various concepts that have been proposed to organize the knowledge about the structure and use of an SLA. One such concept is that an SLA should be based on results rather than on effort. For example, in many instances, attempts are made to describe the results of an IS/IT service in terms of quality levels to be achieved for the IS/IT service, and with the expectation that this is to become the basis for assessing whether or not the terms of the SLA have been met. This agreement is called a results-based SLA, and is an approach which requires that a consensus be reached on the quality levels and that some quantitative measure be assigned to every service and product. IS/IT organi-

zations, including some outsourcers, are known to avoid this path and claim all sorts of technical hurdles to the implementation of this type of agreement. It has been observed in practice that where there is no consensus on a proper description of results; the fall-back position is to describe only the effort and costs to be spent on maintaining the software. These are called effort-based SLAs. However, effort-based SLAs cannot be easily controlled.

Another concept proposed by Bouman is an inventory of guideline templates to be used for the specification of an SLA document between two parties. One should not underestimate the difficulty of establishing service measurement. Additionally, the SLA should clarify the expectations/requirements of each service. In the context of an internal agreement on software maintenance services between the maintenance organization and its users/customers, two attitudes have been reported [April 2001]:

1. On the one hand, the customer wants to concentrate on his business and expects a homogeneous service from his IS/IT organization. The result of this homogeneous service for the customer is the ability to work with a set of information systems without any disturbance whatever from the source of the failure. The way in which this service, these systems, or their infrastructure are constituted is of less interest to the customer, who would like this vision to be reflected in his SLA. This means that a result-based SLA on the total service (including help desk and computer operations) should be described, and not only on the individual/partial IS/IT components (i.e., a server, a network, or software).
2. On the other hand, software maintainers only own a portion of the service as perceived by the customer. They are often quite ready to provide a result-based SLA for their services. But the products are operating on infrastructures that are not under their responsibility (desktop, networks, and platforms), as they only interface with the help desk and computer operations. In [April 2001], the internal SLA of a Cable & Wireless member company is described in detail.

To achieve integrated measurement of all the components of the software requires the involvement of all groups concerned, and requires that someone in IS/IT must own an overall SLA in which all the IS/IT services are described. All the IS/IT service organizations that support the customer must be documented in a unified SLA to satisfy the customer. A unified SLA is one that includes the service levels of all the IS/IT organizations that are involved in supporting the end users. Included are the topics that are more maintenance-specific in a unified SLA:

- Responsibilities of the maintenance customer
- Responsibilities of the maintenance organization
- Description of maintenance services:
  Maintenance program management
  Management of requests, priorities, and the request management software

Corrective, preventive, adaptive, and perfective maintenance
Planning and management of software versions (releases)
Configuration management
License management, escrow delivery, and contracts with third parties
Recovery after disasters
Customer support
Exclusions
Detailed list of supported software, by priority
Service fees
Service hours
Escalation procedures in case of problems
Measures of performance
Reviews and mechanisms for solving disagreements and conflicts

A software maintenance SLA requires a clear definition of service, measurements, and objectives, and needs a documented and detailed inventory of the software that is supported. For each software product, the following topics are to be documented: its customers, specific responsibilities, platforms, support hours, historical work volume, equipment rental and rental customers, availability objectives, parameters of recovery in case of disaster, third-party contracts, and the list of maintainers assigned.

The current approach to SLAs still suffers from a number of problems [Bouman 1999]:

- The customer does not want to be involved and expects homogeneous service from his IS/IT organization. Technical details about services and software products are not very important to him as he is more focused on functionality of the software.
- The description of the expected maintenance results are not written in a way that is readily understood by the customer, but is rather complex and often technical.
- There is confusion in the transfer of responsibilities between the development and maintenance organizations. The concept of the original software warranty is difficult to establish in this context.
- Service rewards and penalties are generally not part of the agreement.

1.11.2 Maintenance Service Contracts—External Service Agreement

Maintainers also require a second type of contract knowledge for contracts involving third parties, who are often contracted to conduct maintenance on software on an annual basis. These contracts are called maintenance service contracts. The soft-

ware industry makes a distinction between a license and a maintenance service contract. In the industry, a maintenance contract will typically include software updates and support activities for use should a problem arise.

A good way to negotiate the maintenance service contract and license price is to act early during the initial purchase of a software. It is often too late if the acquisition or development team has already finalized the initial purchase/project agreement and contracts. The organization that acquires the software is then locked in and there is no supplier/developer incentive to give reductions or additional services for maintenance services. An important rule to remember is that maintainers must be included in the predelivery negotiations in order to use the initial contract/project discussions to ensure that the maintenance-related issues are included clearly and completely, and that maintenance services will provide value for money.

Observing maintenance services contract negotiations between buyers, suppliers, and maintenance managers in the industry makes it possible to better understand the typical maintenance agreement content. Below is an outline for such an agreement:

- Definitions
- Supplier obligations
- Maintenance services and optional services
- Scope of maintenance
- General terms of the agreement
- Customer and customer personnel obligations
- Confidentiality
- Reproduction of documentation and source code
- Limits of responsibility, acts of God, order payments, survival, laws
- Support procedures

The consensus between the two parties regarding the classification of errors and the procedures that they will have to use to report a problem are typically documented in the contract appendices. Maintenance categories, requests for changes, problem severity, response time, and escalation are also typically defined there.

There are still some specific risks inherent to maintenance service contracts:

- The supplier proposes temporary solutions to problems instead of permanent ones.
- The supplier will attempt to negotiate renewals of the license and the maintenance separately in order to make more money on both.
- Additional charges are not clearly defined.
- Escrow services are rarely discussed or included in the costs.

Mastering the maintenance service contract domain requires that a number of maintenance engineers be trained and assigned to software supplier negotiations

(during predelivery activities). Some websites discuss and present software maintenance contract clauses and provide hints that could help businesses keep up with suppliers' business strategies in this area [Business Link 2007].

In summary, the maintenance service contract domain is a specialized one, and it has been the focus of very few publications up to now. The manager and the maintenance engineer are typically at the mercy of the suppliers if they do not acquire some specialized knowledge on how these contracts are structured and negotiated.

1.11.3 Outsourcing Agreements

A third type of service agreement is the software maintenance outsourcing contact. This contract moves the software maintenance to a third party for a period varying from 5 to 10 years. An outsourcing contract is often a global agreement with an IS/IT supplier or an industry leader with an important foothold in the IS/IT sector. It is typically long-term, and once it is in place the organization will sever ties with both its former personnel and the application software itself. In this type of agreement, the outsourcer will take charge of the personnel and will propose service agreement clauses.

The main justifications for outsourcing software maintenance are [Carey 1994]:

- Promises of decreasing costs
- Access to the expertise of the outsourcer's personnel
- Move from a fixed-cost structure to a variable-cost structure
- Collect revenue from the sale of an asset
- IS/IT not being one of the company's strategic activities
- Transfer of technical details and problems to the outsourcer

A typical outsourcing contract is quite different from a service contract. For instance, the outsourcer takes over the application software, as well as the source codes, and treats software packages separately (SAP/R3, Oracle HR, etc.) from the general agreement:

- The outsourcer offers typically a 90-day warranty following the correction of a specific problem. If the problem has not been fixed to the client's satisfaction, the outsourcer provides additional maintenance services free of charge.
- The outsourcer asks the client to determine the priority of maintenance work items.
- The outsourcer keeps a list and a history of problems submitted using help desk software.
- While the ISO 14764 software maintenance work categories are not often used, the following work categories have been often observed: corrections, preventive, regulations, release control, and ad hoc reports.

- A reference to an internal service agreement and measurement is made, but no report, measures, or review process is formally proposed.
- The duration of the outsourcing agreement is between 5 and 10 years.

McCracken [2002] reports that 50% of companies become involved in outsourcing without having a clear service agreement, and he also refers to a report from the Gartner Group that places this figure at 85%.

In summary, outsourcing contracts documents contain financial objectives, but do not yet completely define outsourced services (according to international standards) and targeted service levels (with reports and precise measures).

1.12 SOFTWARE MAINTENANCE BENCHMARKING

A popular definition of benchmarking was originally reported by David T. Kearns, CEO of Xerox: "Benchmarking is the continuous process of measuring products, services and practices against the toughest competitors or those companies recognized as industry leaders."

Three types of benchmarking were proposed by Xerox. The first is internal benchmarking, which aims to compare different sectors in the organization. For software maintenance, an internal comparison allows one to draw conclusions for improvement. The second type, competitive benchmarking, requires data from direct competitors. This approach is made difficult by problems with accessing information from other organizations and, thus, understanding how these competitors perform in their chosen functional area. The last proposed approach is functional benchmarking, which consists in finding organizations that perform well in the same functional area, but in other industries. For general maintenance of hardware equipment, the methods used by NASA, nuclear power plants, and the aerospace industry can be examined to find useful information.

What about the functional benchmarking of software? A prerequisite for carrying out a benchmarking exercise is a clear understanding of internal processes. Although it is not necessary to have a very elaborate measurement program in place, there must be data on hand. Some argue that benchmarking could be a waste of time if the organization does not possess that understanding and a certain number of internal process measurements.

Two approaches are more common in the software industry. The approach most often observed is competitive benchmarking with the participation of an outside benchmarking service provider specialized in computer services benchmarking. For example, some benchmarking organizations use their own commercial approach to collect and analyze data, based on their own corporate questionnaire for data collection. A few weaknesses were noted in this approach by the managers we interviewed on the topic:

- Short time period (2 months) and difficulty of collecting validated data (often the requested data was not available and had to be made up on the fly)

- Comparative data is hidden (it is unknown against whom the comparison is being made)
- Problems with the quality of data (e.g., quality of the count of the lines of code)
- Credibility of the lines of code to function points conversion ratios
- Number of resulting diagrams/graphs (40 comparative viewpoints)
- The interpretation of results does not take into account specific facts but is rather made with generalizations that are hard to contradict or support
- Difficult to get detailed information about the best performers and what they do to get this kind of performance

In summary, a good number of measurements are presented to the organization, but it is difficult to know whether or not they are valid, or to find out the practices of those companies that claim to perform better.

A number of risks of the same type have been reported. Firms that use this approach specifically for maintenance present, at the end of the exercise, variants of the following graphs/measures:

- Function points supported per person (in-house development)
- Function points supported per person (in-house development + software packages)
- Cost by supported function points
- Average age (in years) of application software (currently in production)
- Age groups (% by functionality) of application software (currently in production)
- Number of supported programming languages
- Supported data structures (sequential, indexed, relational, others)
- % of programming types (maintenance, improvements, new applications)
- Support—environment complexity index (based on the number of users, data size, platform size, and power)
- Support—technical diversity (number of applications, programming languages, data archiving technologies, tools, and operating systems)
- Support—use of CASE tools
- % of personnel stability (turnover rate)
- Duration of employment (years)
- Human resources level (% of total company employees)
- Salaries
- Training effort (number of training days per person per year)
- Ratio of faults in production (by criticality: critical, major, minor, or cosmetic) per 1000 supported FPs
- Ratio of faults in production (by cause category: design, programming, environment, or other) per 1000 supported FPs

A second approach is benchmarking within a user group. The best-known ones are the IT Benchmarking User Group and the International Software Benchmarking Standards Group [ISBSG 2007]. The advantage of these organizations is that they, under some conditions, allow their members access to their databases and thus enable the sharing of experiences. Participation in this type of activity is specialized and does not include the purchase of a "ready-to-use" service. Therefore, only a few organizations will participate in these user groups because they require a local champion dedicated to this task for a number of years. It has been, however, observed that up until now such organizations had little data in software maintenance but rather focused on software development projects.

Capers Jones mentions a last kind of benchmarking which he calls "Assessment Benchmarking," which he compares to a medical evaluation of the IT organization in order to establish a diagnostic on what works and what does not using an industry guide of best practices [Jones 1994]. The SEI's model is the most widely known and used for this type of benchmarking. In the software industry, this exercise is considered as a process improvement exercise rather than as a benchmarking exercise.

In summary, software benchmarking does not easily allow scientific comparison because the data is neither controlled nor validated. However, benchmarking is sometimes done too quickly and the conclusions do not allow for the clear identification of lessons for improvement from benchmarked organizations. Benchmarking activities with open databases are now feasible but require sustained help from a knowledgeable champion. Organizations in other, often more mature, industries publish insightful results from the use of this benchmarking practice.

1.13 SUMMARY

In this chapter, many maintenance issues have been presented from two perspectives (internal and external). Maintainers are aware that customers perceive these issues through quality of service and costs, both of which are key drivers of customer satisfaction.

Also presented was the perception of software maintenance engineers and managers, documenting the reasons why they claim it is so difficult to maintain software. A large number of these problems are management related and occur in a constantly changing environment. Moreover, many of the difficulties encountered in maintenance have their origin in the initial development of the software. This highlights the importance of involvement in predelivery and transition activities; with other partners involved, maintainers must be able to actively participate in early project and contract negotiations to ensure that their voice is heard.

Software maintenance is a specific domain of software engineering and is now included in the SWEBOK as a main knowledge area of software engineering.

Two standards were central to software maintenance for many years (ISO 14764 and IEEE 1219). A project within ISO JTC1/SC7 was done in 2005–2006 to harmo-

nize the two standards. In 2006, a new version of ISO 14764 was published and IEEE1219 was withdrawn. However, it is important to note that international standards do not include all the maintenance activities observed in the industry today and inventoried in many recent publications.

Software maintenance and software development have indeed some activities in common. Maintenance refers to specific software development activities, listed clearly in international standards. The standards also indicate that maintainers must make sure to adapt them to the specific maintenance context and not just use them as the developers intended.

This chapter has also highlighted that measurement plays an important role in assessing maintenance process and product quality. Maintainers have specific processes, as well as specific measurement programs and approaches. Products can also be assessed for their quality by specifying maintainability measures (external and internal maintainability).

It has further been demonstrated that maintainers need contract knowledge in order to ensure that they can influence software quality, build internal agreements with their customers, and manage actively their suppliers.

To conclude, software maintenance is a unique domain and a maturity model that includes the specific characteristics of maintenance is worth designing.

1.14 EXERCISES

1. You have been promoted to the position of software maintenance manager. Meanwhile, there is a heated discussion about developer/maintainer interface problems. The development manager is lobbying to bring this issue under his control and, in this way, put an end to these problems which impact clients without any obvious benefits. You must prepare to discuss this situation at a senior management meeting that will be attended by your colleagues in software development. Draw up your own proposal and present the pros and cons of your colleagues' position.
2. Describe the unique characteristics of software maintenance.
3. It has been pointed out to you that, over time, the development teams have established fine-tuned development processes, and that it would be economical and useful to reuse these processes for maintenance purposes. In which standards can you find pointers about development processes that can be of use in software maintenance? Describe your proposed strategy for applying these standards to maintenance, and use them to support your case.
4. Your organization's position is that maintenance activities belong at the end of the software development life cycle. However, this contention contributes to marginalizing your work, and does not fully represent all the maintenance you perform. What changes can you propose to the software life-cycle documentation to provide a more comprehensive view of soft-

ware maintenance activities? What standards could be of use to you for this purpose?

5. There are two of you working in software maintenance. Your boss requires that you follow ISO standards. How can you achieve this without spending all your weekends at work?
6. The standards do not cover all the maintenance activities performed in your organization. Describe five activities that do not appear in ISO 14764. Document the source of your information and explain why, in your opinion, these activities are not included in this international standard.
7. Operations services phoned you this morning and asked you to carry out checks to ensure that the software will function correctly on the new network. Explain (1) in which maintenance category you classify this work, (2) what kinds of tests are necessary, and (3) do you have to request authorization from your customers before starting work?
8. This week, your customer sends you a third request that entails a significant modification. He is new to his job. Explain to him, in simple and clear terms, the differences between software maintenance and software development.
9. Your boss tells you that, during the last negotiation meeting with the customers, they did not accept the proposed increases in service charges. Help your boss by proposing an alternative solution based on the various maintenance categories.
10. Describe some types of preventive requests in software maintenance.