

CHAPTER 6

SOFTWARE MAINTENANCE

ACRONYMS

CMMI	Capability Maturity Model Integration
ICSM	International Conference on Software Maintenance
SCM	Software Configuration Management
SQA	Software Quality Assurance
V&V	Verification and Validation
Y2K	Year 2000

INTRODUCTION

Software development efforts result in the delivery of a software product which satisfies user requirements. Accordingly, the software product must change or evolve. Once in operation, defects are uncovered, operating environments change, and new user requirements surface. The maintenance phase of the life cycle begins following a warranty period or post-implementation support delivery, but maintenance activities occur much earlier.

Software maintenance is an integral part of a software life cycle. However, it has not, historically, received the same degree of attention that the other phases have. Historically, software development has had a much higher profile than software maintenance in most organizations. This is now changing, as organizations strive to squeeze the most out of their software development investment by keeping software operating as long as possible. Concerns about the Year 2000 (Y2K) rollover focused significant attention on the software maintenance phase, and the Open Source paradigm has brought further attention to the issue of maintaining software artifacts developed by others.

In the Guide, software maintenance is defined as the totality of activities required to provide cost-effective support to software. Activities are performed during the pre-delivery stage, as well as during the post-delivery stage. Pre-delivery activities include planning for post-delivery operations, for maintainability, and for logistics determination for transition activities. Post-delivery activities include software modification, training, and operating or interfacing to a help desk.

The Software Maintenance KA is related to all other aspects of software engineering. Therefore, this KA description is linked to all other chapters of the Guide.

BREAKDOWN OF TOPICS FOR SOFTWARE MAINTENANCE

The Software Maintenance KA breakdown of topics is shown in Figure 1.

1. Software Maintenance Fundamentals

This first section introduces the concepts and terminology that form an underlying basis to understanding the role and scope of software maintenance. The topics provide definitions and emphasize why there is a need for maintenance. Categories of software maintenance are critical to understanding its underlying meaning.

1.1. Definitions and Terminology

[IEEE1219-98:s3.1.1.12; IEEE12207.0-96:s3.1,s5.5; ISO14764-99:s6.1]

Software maintenance is defined in the IEEE Standard for Software Maintenance, IEEE 1219, as 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. The standard also addresses maintenance activities prior to delivery of the software product, but only in an information appendix of the standard.

The IEEE/EIA 12207 standard for software life cycle processes essentially depicts maintenance as one of the primary life cycle processes, and describes maintenance as the process of a software product undergoing “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.” ISO/IEC 14764, the international standard for software maintenance, defines software maintenance in the same terms as IEEE/EIA 12207 and emphasizes the pre-delivery aspects of maintenance, planning, for example.

1.2. Nature of Maintenance

[Pfl01:c11s11.2]

Software maintenance sustains the software product throughout its operational life cycle. Modification requests are logged and tracked, the impact of proposed changes is determined, code and other software artifacts are modified, testing is conducted, and a new version of the software product is released. Also, training and daily support are provided to users. Pfleeger [Pfl01] states that “maintenance has a broader scope, with more to track and control” than development.

A maintainer is defined by IEEE/EIA 12207 as an organization which performs maintenance activities [IEEE12207.0-96]. In this KA, the term will sometimes refer to individuals who perform those activities, contrasting them with the developers.

IEEE/EIA 12207 identifies the primary activities of software maintenance as: process implementation; problem and modification analysis; modification implementation; maintenance review/acceptance; migration; and retirement. These activities are discussed in topic 3.2 *Maintenance Activities*.

Maintainers can learn from the developer’s knowledge of the software. Contact with the developers and early involvement by the maintainer helps reduce the maintenance effort. In some instances, the software engineer cannot be reached or has moved on to other tasks, which creates an additional challenge for the maintainers. Maintenance must take the products of the development, code, or documentation, for example, and support them immediately and evolve/maintain them progressively over the software life cycle.

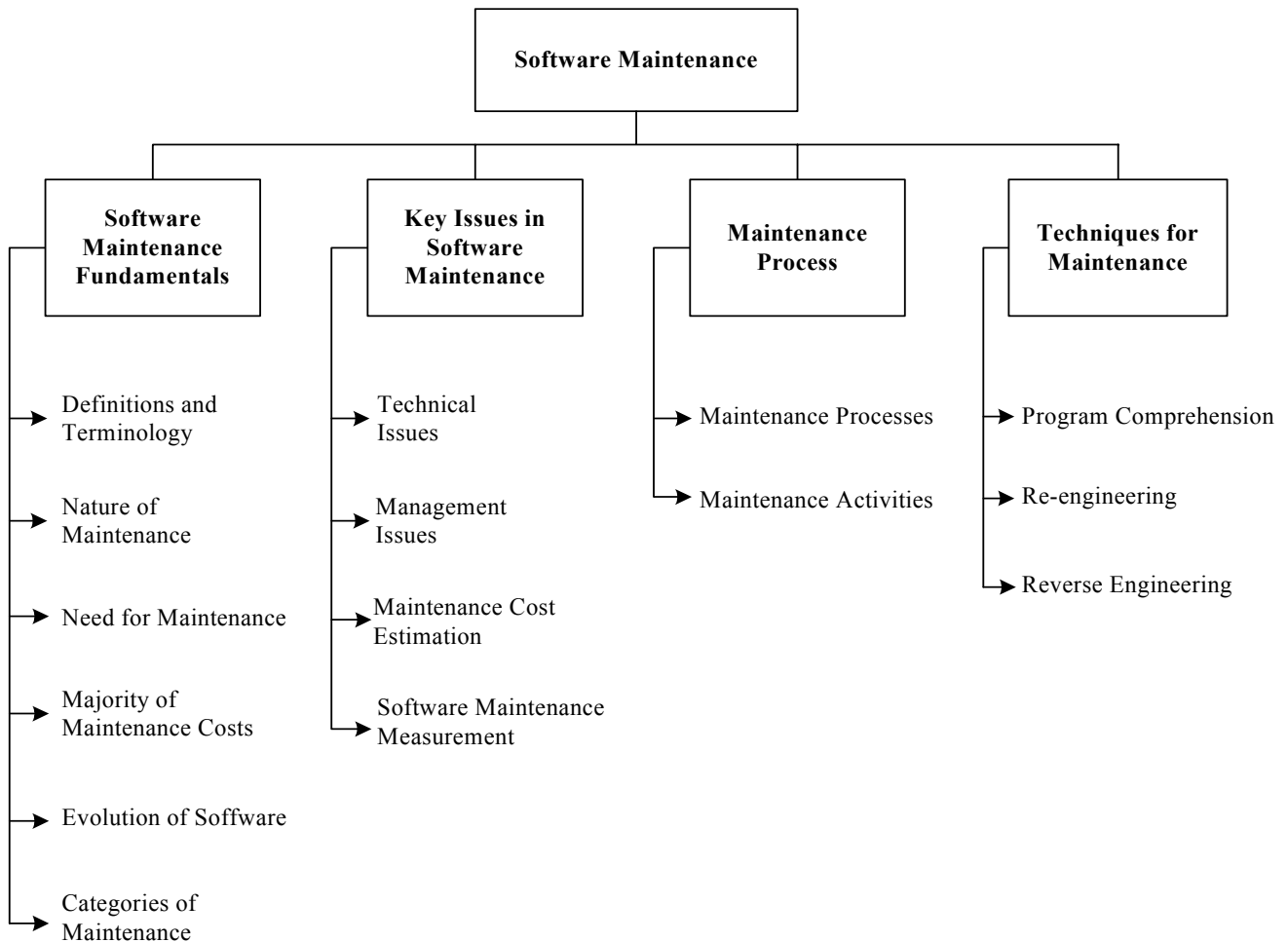


Figure 1 Breakdown of topics for the Software Maintenance KA

1.3. Need for Maintenance

[Pfl01:c11s11.2; Pig97: c2s2.3; Tak97:c1]

Maintenance is needed to ensure that the software continues to satisfy user requirements. Maintenance is applicable to software developed using any software life cycle model (for example, spiral). The system changes due

to corrective and non-corrective software actions. Maintenance must be performed in order to:

- ♦ Correct faults
- ♦ Improve the design
- ♦ Implement enhancements

- ◆ Interface with other systems
- ◆ Adapt programs so that different hardware, software, system features, and telecommunications facilities can be used
- ◆ Migrate legacy software
- ◆ Retire software

The maintainer's activities comprise four key characteristics, according to Pfleeger [Pfl01]:

- ◆ Maintaining control over the software's day-to-day functions
- ◆ Maintaining control over software modification
- ◆ Perfecting existing functions
- ◆ Preventing software performance from degrading to unacceptable levels

1.4. Majority of Maintenance Costs

[Abr93:63-90; Pfl01:c11s11.3; Pig97:c3; Pre01:c30s2.1,c30s2.2]

Maintenance consumes a major share of software life cycle financial resources. A common perception of software maintenance is that it merely fixes faults. However, studies and surveys over the years have indicated that the majority, over 80%, of the software maintenance effort is used for non-corrective actions. [Abr93, Pig97, Pre01] Jones (Jon91) describes the way in which software maintenance managers often group enhancements and corrections together in their management reports. This inclusion of enhancement requests with problem reports contributes to some of the misconceptions regarding the high cost of corrections. Understanding the categories of software maintenance helps to understand the structure of software maintenance costs. Also, understanding the factors that influence the maintainability of a system can help to contain costs. Pfleeger [Pfl01] presents some of the technical and non-technical factors affecting software maintenance costs, as follows:

- ◆ Application type
- ◆ Software novelty
- ◆ Software maintenance staff availability
- ◆ Software life span
- ◆ Hardware characteristics
- ◆ Quality of software design, construction, documentation and testing

1.5. Evolution of Software

[Art88:c1s1.0,s1.1,s1.2,c11s1.1,s1.2; Leh97:108-124], (Bel72)

Lehman first addressed software maintenance and evolution of systems in 1969. Over a period of twenty years, his research led to the formulation of eight "Laws of Evolution". [Leh97] Key findings include the fact that maintenance is evolutionary developments, and that

maintenance decisions are aided by understanding what happens to systems (and software) over time. Others state that maintenance is continued development, except that there is an extra input (or constraint)—existing large software is never complete and continues to evolve. As it evolves, it grows more complex unless some action is taken to reduce this complexity.

Since software demonstrates regular behavior and trends, these can be measured. Attempts to develop predictive models to estimate maintenance effort have been made, and, as a result, useful management tools have been developed. [Art88], (Bel72)

1.6. Categories of Maintenance

[Art88:c1s1.2; Lie78; Dor02:v1c9s1.5; IEEE1219-98:s3.1.1,s3.1.2,s3.1.7,A.1.7; ISO14764-99:s4.1,s4.3,s4.10, s4.11,s6.2; Pig97:c2s2.3]

Lientz & Swanson initially defined three categories of maintenance: corrective, adaptive, and perfective. [Lie78; IEEE1219-98] This definition was later updated in the Standard for Software Engineering-Software Maintenance, ISO/IEC 14764 to include four categories, as follows:

- ◆ Corrective maintenance: Reactive modification of a software product performed after delivery to correct discovered problems
- ◆ Adaptive maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment
- ◆ Perfective maintenance: Modification of a software product after delivery to improve performance or maintainability
- ◆ Preventive maintenance: Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults

ISO/IEC 14764 classifies adaptive and perfective maintenance as enhancements. It also groups together the corrective and preventive maintenance categories into a correction category, as shown in Table 1. Preventive maintenance, the newest category, is most often performed on software products where safety is critical.

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

Table 1: Software maintenance categories

2. Key Issues in Software Maintenance

A number of key issues must be dealt with to ensure the effective maintenance of software. It is important to

understand that software maintenance provides unique technical and management challenges for software engineers. Trying to find a fault in software containing 500K lines of code that the software engineer did not develop is a good example. Similarly, competing with software developers for resources is a constant battle. Planning for a future release, while coding the next release and sending out emergency patches for the current release, also creates a challenge. The following section presents some of the technical and management issues related to software maintenance. They have been grouped under the following topic headings:

- ◆ Technical issues
- ◆ Management issues
- ◆ Cost estimation and
- ◆ Measures

2.1. Technical Issues

2.1.1. Limited understanding

[Dor02:v1c9s1.11.4; Pfl01:c11s11.3; Tak97:c3]

Limited understanding refers to how quickly a software engineer can understand where to make a change or a correction in software which this individual did not develop. Research indicates that some 40% to 60% of the maintenance effort is devoted to understanding the software to be modified. Thus, the topic of software comprehension is of great interest to software engineers. Comprehension is more difficult in text-oriented representation, in source code, for example, where it is often difficult to trace the evolution of software through its releases/versions if changes are not documented and when the developers are not available to explain it, which is often the case. Thus, software engineers may initially have a limited understanding of the software, and much has to be done to remedy this.

2.1.2. Testing

[Art88:c9; Pfl01:c11s11.3]

The cost of repeating full testing on a major piece of software can be significant in terms of time and money. Regression testing, the selective retesting of a software or component to verify that the modifications have not caused unintended effects, is important to maintenance. As well, finding time to test is often difficult. There is also the challenge of coordinating tests when different members of the maintenance team are working on different problems at the same time. [Pfl01] When software performs critical functions, it may be impossible to bring it offline to test. The Software Testing KA provides additional information and references on the matter in its sub-topic 2.2.6 Regression testing.

2.1.3. Impact analysis

[Art88:c3; Dor02:v1c9s1.10; Pfl01: c11s11.5]

Impact analysis describes how to conduct, cost effectively, a complete analysis of the impact of a change in existing software. Maintainers must possess an intimate knowledge of the software's structure and content [Pfl01]. They use that knowledge to perform impact analysis, which identifies all systems and software products affected by a software change request and develops an estimate of the resources needed to accomplish the change. [Art88] Additionally, the risk of making the change is determined. The change request, sometimes called a modification request (MR) and often called a problem report (PR), must first be analyzed and translated into software terms. [Dor02] It is performed after a change request enters the software configuration management process. Arthur [Art88] states that the objectives of impact analysis are:

- ◆ Determination of the scope of a change in order to plan and implement work
- ◆ Development of accurate estimates of resources needed to perform the work
- ◆ Analysis of the cost/benefits of the requested change
- ◆ Communication to others of the complexity of a given change

The severity of a problem is often used to decide how and when a problem will be fixed. The software engineer then identifies the affected components. Several potential solutions are provided and then a recommendation is made as to the best course of action.

Software designed with maintainability in mind greatly facilitates impact analysis. More information can be found in the Software Configuration Management KA.

2.1.4. Maintainability

[ISO14764-99:s6.8s6.8.1; Pfl01: c9s9.4; Pig97:c16]

How does one promote and follow up on maintainability issues during development? The IEEE [IEEE610.12-90] defines maintainability as the ease with which software can be maintained, enhanced, adapted, or corrected to satisfy specified requirements. ISO/IEC defines maintainability as one of the quality characteristics (ISO9126-01).

Maintainability sub-characteristics must be specified, reviewed, and controlled during the software development activities in order to reduce maintenance costs. If this is done successfully, the maintainability of the software will improve. This is often difficult to achieve because the maintainability sub-characteristics are not an important focus during the software development process. The developers are preoccupied with many other things and often disregard the maintainer's requirements. This in turn can, and often does, result in a lack of system documentation, which is a leading cause of difficulties in program comprehension and impact analysis. It has also been observed that the presence of systematic and mature

processes, techniques, and tools helps to enhance the maintainability of a system.

2.2. Management Issues

2.2.1. Alignment with organizational objectives [Ben00:c6sa; Dor02:v1c9s1.6]

Organizational objectives describe how to demonstrate the return on investment of software maintenance activities. Bennett [Ben00] states that “initial software development is usually project-based, with a defined time scale and budget. The main emphasis is to deliver on time and within budget to meet user needs. In contrast, software maintenance often has the objective of extending the life of software for as long as possible. In addition, it may be driven by the need to meet user demand for software updates and enhancements. In both cases, the return on investment is much less clear, so that the view at senior management level is often of a major activity consuming significant resources with no clear quantifiable benefit for the organization.”

2.2.2. Staffing [Dek92:10-17; Dor02:v1c9s1.6; Par86: c4s8-c4s11] (Lie81)

Staffing refers to how to attract and keep software maintenance staff. Maintenance is often not viewed as glamorous work. Deklava provides a list of staffing-related problems based on survey data. [Dek92] As a result, software maintenance personnel are frequently viewed as “second-class citizens” (Lie81) and morale therefore suffers. [Dor02]

2.2.3. Process [Pau93; Ben00:c6sb; Dor02:v1c9s1.3]

Software process is a set of activities, methods, practices, and transformations which people use to develop and maintain software and the associated products. [Pau93] At the process level, software maintenance activities share much in common with software development (for example, software configuration management is a crucial activity in both). [Ben00] Maintenance also requires several activities which are not found in software development (see section 3.2 on unique activities for details). These activities present challenges to management. [Dor02]

2.2.4. Organizational aspects of maintenance [Pfl01:c12s12.1-c12s12.3; Par86:c4s7; Pig97:c2s2.5; Tak97:c8]

Organizational aspects describe how to identify which organization and/or function will be responsible for the maintenance of software. The team that develops the software is not necessarily assigned to maintain the software once it is operational.

In deciding where the software maintenance function will be located, software engineering organizations may, for example, stay with the original developer or go to a separate team (or maintainer). Often, the maintainer option

is chosen to ensure that the software runs properly and evolves to satisfy changing user needs. Since there are many pros and cons to each of these options [Par86, Pig97], the decision should be made on a case-by-case basis. What is important is the delegation or assignment of the maintenance responsibility to a single group or person [Pig97], regardless of the organization’s structure.

2.2.5. Outsourcing [Dor02:v1c9s1.7; Pig97:c9s9.1,s9.2], (Car94; McC02)

Outsourcing of maintenance is becoming a major industry. Large corporations are outsourcing entire portfolios of software systems, including software maintenance. More often, the outsourcing option is selected for less mission-critical software, as companies are unwilling to lose control of the software used in their core business. Carey (Car94) reports that some will outsource only if they can find ways of maintaining strategic control. However, control measures are hard to find. One of the major challenges for the outsourcers is to determine the scope of the maintenance services required and the contractual details. McCracken (McC02) states that 50% of outsourcers provide services without any clear service-level agreement. Outsourcing companies typically spend a number of months assessing the software before they will enter into a contractual relationship. [Dor02] Another challenge identified is the transition of the software to the outsourcer. [Pig97]

2.3. Maintenance Cost Estimation

Software engineers must understand the different categories of software maintenance, discussed above, in order to address the question of estimating the cost of software maintenance. For planning purposes, estimating costs is an important aspect of software maintenance.

2.3.1. Cost estimation [Art88:c3; Boe81:c30; Jon98:c27; Pfl01:c11s11.3; Pig97:c8]

It was mentioned in sub-topic 2.1.3, Impact Analysis, that impact analysis identifies all systems and software products affected by a software change request and develops an estimate of the resources needed to accomplish that change. [Art88]

Maintenance cost estimates are affected by many technical and non-technical factors. ISO/IEC14764 states that “the two most popular approaches to estimating resources for software maintenance are the use of parametric models and the use of experience” [ISO14764-99:s7.4.1]. Most often, a combination of these is used.

2.3.2. Parametric models [Ben00:s7; Boe81:c30; Jon98:c27; Pfl01:c11s11.3]

Some work has been undertaken in applying parametric cost modeling to software maintenance. [Boe81, Ben00] Of

significance is that data from past projects are needed in order to use the models. Jones [Jon98] discusses all aspects of estimating costs, including function points (IEEE14143.1-00), and provides a detailed chapter on maintenance estimation.

2.3.3. Experience

[ISO14764-00:s7,s7.2,s7.2.1,s7.2.4; Pig97:c8; Sta94]

Experience, in the form of expert judgment (using the Delphi technique, for example), analogies, and a work breakdown structure, are several approaches which should be used to augment data from parametric models. Clearly the best approach to maintenance estimation is to combine empirical data and experience. These data should be provided as a result of a measurement program.

2.4. Software Maintenance Measurement

[IEEE1061-98:A.2; Pig97:c14s14.6; Gra87 ; Tak97: c6s6.1-c6s6.3]

Grady and Caswell [Gra87] discuss establishing a corporate-wide software measurement program, in which software maintenance measurement forms and data collection are described. The Practical Software and Systems Measurement (PSM) project describes an issue-driven measurement process that is used by many organizations and is quite practical. [McG01]

There are software measures that are common to all endeavors, the following categories of which the Software Engineering Institute (SEI) has identified: size; effort; schedule; and quality. [Pig97] These measures constitute a good starting point for the maintainer. Discussion of process and product measurement is presented in the Software Engineering Process KA. The software measurement program is described in the Software Engineering Management KA.

2.4.1. Specific Measures

[Car90:s2-s3; IEEE1219-98:Table3; Sta94:p239-249]

Abran [Abr93] presents internal benchmarking techniques to compare different internal maintenance organizations. The maintainer must determine which measures are appropriate for the organization in question. [IEEE1219-98; ISO9126-01; Sta94] suggests measures which are more specific to software maintenance measurement programs.

That list includes a number of measures for each of the four sub-characteristics of maintainability:

- ♦ *Analyzability*: Measures of the maintainer's effort or resources expended in trying to diagnose deficiencies or causes of failure, or in identifying parts to be modified
- ♦ *Changeability*: Measures of the maintainer's effort associated with implementing a specified modification
- ♦ *Stability*: Measures of the unexpected behavior of software, including that encountered during testing
- ♦ *Testability*: Measures of the maintainer's and users' effort in trying to test the modified software

Certain measures of the maintainability of software can be obtained using available commercial tools. (Lag96; Apr00)

3. Maintenance Process

The *Maintenance Process* subarea provides references and standards used to implement the software maintenance process. The *Maintenance Activities* topic differentiates maintenance from development and shows its relationship to other software engineering activities.

The need for software engineering process is well documented. CMMI® models apply to software maintenance processes, and are similar to the developers' processes. [SEI01] Software Maintenance Capability Maturity models which address the unique processes of software maintenance are described in (Apr03, Nie02, Kaj01).

3.1. Maintenance Processes

[IEEE1219-98:s4; ISO14764-99:s8; IEEE12207.0-96:s5.5; Par86:c7s1; Pig97:c5; Tak97:c2]

Maintenance processes provide needed activities and detailed inputs/outputs to those activities, and are described in software maintenance standards IEEE 1219 and ISO/IEC 14764.

The maintenance process model described in the Standard for Software Maintenance (IEEE1219) starts with the software maintenance effort during the post-delivery stage and discusses items such as planning for maintenance. That process is depicted in Figure 2.

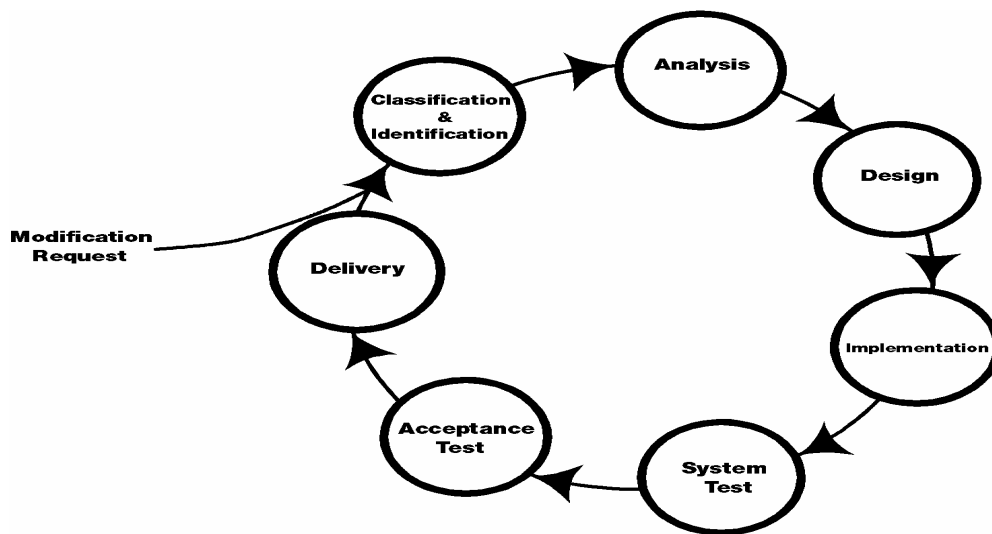


Figure 2 The IEEE1219-98 Maintenance Process Activities

ISO/IEC 14764 [ISO14764-99] is an elaboration of the IEEE/EIA 12207.0-96 maintenance process. The activities of the ISO/IEC maintenance process are similar to those of the IEEE, except that they are aggregated a little differently. The maintenance process activities developed by ISO/IEC are shown in Figure 3.

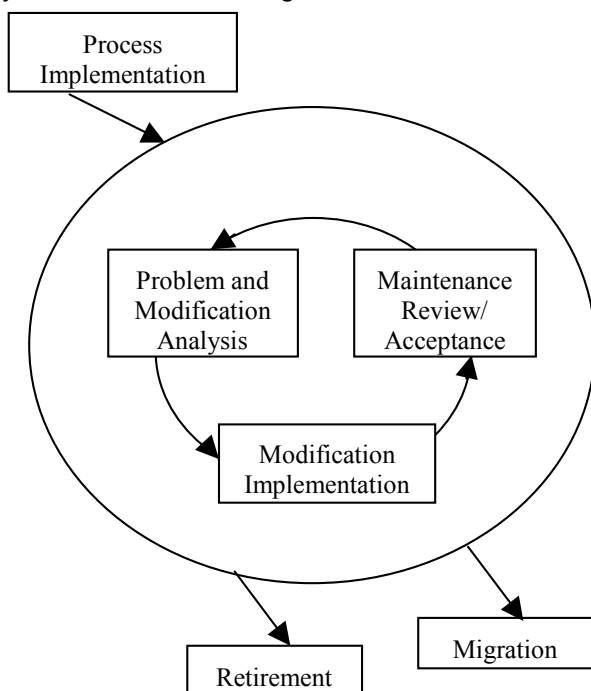


Figure 3 ISO/IEC 14764-00 Software Maintenance Process

Each of the ISO/IEC 14764 primary software maintenance activities is further broken down into tasks, as follows.

- ♦ Process Implementation
- ♦ Problem and Modification Analysis
- ♦ Modification Implementation
- ♦ Maintenance Review/Acceptance
- ♦ Migration
- ♦ Software Retirement

Takang & Grubb [Tak97] provide a history of maintenance process models leading up to the development of the IEEE and ISO/IEC process models. Parikh [Par86] also gives a good overview of a generic maintenance process. Recently, agile methodologies have been emerging which promote light processes. This requirement emerges from the ever-increasing demand for fast turn-around of maintenance services. Some experiments with Extreme maintenance are presented in (Poo01).

3.2. Maintenance Activities

As already noted, many maintenance activities are similar to those of software development. Maintainers perform analysis, design, coding, testing, and documentation. They must track requirements in their activities just as is done in development, and update documentation as baselines change. ISO/IEC14764 recommends that, when a maintainer refers to a similar development process, he must adapt it to meet his specific needs [ISO14764-99:s8.3.2.1, 2]. However, for software maintenance, some activities involve processes unique to software maintenance.

3.2.1. Unique activities

[Art88:c3; Dor02:v1c9s1.9.1; IEEE1219-98:s4.1,s4.2; ISO14764-99:s8.2.2.1,s8.3.2.1; Pfl01:c11s11.2]

There are a number of processes, activities, and practices that are unique to software maintenance, for example:

- ♦ Transition: a controlled and coordinated sequence of activities during which software is transferred progressively from the developer to the maintainer [Dek92, Pig97]
- ♦ Modification Request Acceptance/Rejection: modification request work over a certain size/effort/complexity may be rejected by maintainers and rerouted to a developer [Dor02], (Apr01)
- ♦ Modification Request and Problem Report Help Desk: an end-user support function that triggers the assessment, prioritization, and costing of modification requests [Ben00]
- ♦ Impact Analysis (see section 2.1.3 for details)
- ♦ Software Support: help and advice to users concerning a request for information (for example, business rules, validation, data meaning and ad-hoc requests/reports) (Apr03)
- ♦ Service Level Agreements (SLAs) and specialized (domain-specific) maintenance contracts which are the responsibility of the maintainers (Apr01)

3.2.2. Supporting activities

[IEEE1219-98:A.7,A.11; IEEE12207.0-96:c6,c7; ITI01; Pig97:c10s10.2,c18] ;(Kaj01)

Maintainers may also perform supporting activities, such as software maintenance planning, software configuration management, verification and validation, software quality assurance, reviews, audits, and user training.

Another supporting activity, maintainer training, is also needed. [Pig97; IEEE12207.0-96] (Kaj01)

3.2.3. Maintenance planning activity

[IEEE1219-98:A.3; ISO14764-99:s7; ITI01; Pig97:c7,c8]

An important activity for software maintenance is planning, and maintainers must address the issues associated with a number of planning perspectives:

- ♦ Business planning (organizational level)
- ♦ Maintenance planning (transition level)
- ♦ Release/version planning (software level)
- ♦ Individual software change request planning (request level)

At the individual request level, planning is carried out during the impact analysis (refer to sub-topic 2.1.3 Impact Analysis for details). The release/version planning activity requires that the maintainer [ITI01]:

- ♦ Collect the dates of availability of individual requests
- ♦ Agree with users on the content of subsequent releases/versions
- ♦ Identify potential conflicts and develop alternatives

- ♦ Assess the risk of a given release and develop a back-out plan in case problems should arise
- ♦ Inform all the stakeholders

Whereas software development projects can typically last from some months to a few of years, the maintenance phase usually lasts for many years. Making estimates of resources is a key element of maintenance planning. Those resources should be included in the developers' project planning budgets. Software maintenance planning should begin with the decision to develop a new system and should consider quality objectives (IEEE1061-98). A concept document should be developed, followed by a maintenance plan.

The concept document for maintenance [ISO14764-99:s7.2] should address:

- ♦ The scope of the software maintenance
- ♦ Adaptation of the software maintenance process
- ♦ Identification of the software maintenance organization
- ♦ An estimate of software maintenance costs

The next step is to develop a corresponding software maintenance plan. This plan should be prepared during software development, and should specify how users will request software modifications or report problems. Software maintenance planning [Pig97] is addressed in IEEE 1219 [IEEE1219-98] and ISO/IEC 14764. [ISO14764-99] ISO/IEC14764 provides guidelines for a maintenance plan.

Finally, at the highest level, the maintenance organization will have to conduct business planning activities (budgetary, financial, and human resources) just like all the other divisions of the organization. The management knowledge required to do so can be found in the Related Disciplines of Software Engineering chapter.

3.2.4. Software configuration management

[Art88:c2,c10; IEEE1219-98:A.11; IEEE12207.0-96:s6.2; Pfl01:c11s11.5; Tak97:c7]

The IEEE Standard for Software Maintenance, IEEE 1219 [IEEE1219-98], describes software configuration management as a critical element of the maintenance process. Software configuration management procedures should provide for the verification, validation, and audit of each step required to identify, authorize, implement, and release the software product.

It is not sufficient to simply track Modification Requests or Problem Reports. The software product and any changes made to it must be controlled. This control is established by implementing and enforcing an approved software configuration management (SCM) process. The Software Configuration Management KA provides details of SCM and discusses the process by which software change requests are submitted, evaluated, and approved. SCM for software maintenance is different from SCM for software

development in the number of small changes that must be controlled on operational software. The SCM process is implemented by developing and following a configuration management plan and operating procedures. Maintainers participate in Configuration Control Boards to determine the content of the next release/version.

3.2.5. Software quality

[Art98:c7s4; IEEE12207.0-96:s6.3; IEEE1219-98:A.7; ISO14764-99:s5.5.3.2]

It is not sufficient, either, to simply hope that increased quality will result from the maintenance of software. It must be planned and processes implemented to support the maintenance process. The activities and techniques for Software Quality Assurance (SQA), V&V, reviews, and audits must be selected in concert with all the other processes to achieve the desired level of quality. It is also recommended that the maintainer adapt the software development processes, techniques and deliverables, for instance testing documentation, and test results. [ISO14764-99]

More details can be found in the Software Quality KA.

4. Techniques for Maintenance

This subarea introduces some of the generally accepted techniques used in software maintenance.

4.1. Program Comprehension

[Arn92:c14; Dor02:v1c9s1.11.4; Tak97:c3]

Programmers spend considerable time in reading and understanding programs in order to implement changes. Code browsers are key tools for program comprehension. Clear and concise documentation can aid in program comprehension.

4.2. Reengineering

[Arn92:c1,c3-c6; Dor02:v1c9s1.11.4; IEEE1219-98:B.2], (Fow99)

Reengineering is defined as the examination and alteration of software to reconstitute it in a new form, and includes the subsequent implementation of the new form. Dorfman and Thayer [Dor02] state that reengineering is the most radical (and expensive) form of alteration. Others believe that reengineering can be used for minor changes. It is often not undertaken to improve maintainability, but to replace aging legacy software. Arnold [Arn92] provides a comprehensive compendium of topics, for example: concepts, tools and techniques, case studies, and risks and benefits associated with reengineering.

4.3. Reverse engineering

[Arn92:c12; Dor02:v1c9s1.11.3; IEEE1219-98:B.3; Tak97:c4, Hen01]

Reverse engineering is the process of analyzing software to identify the software's components and their inter-relationships and to create representations of the software in another form or at higher levels of abstraction. Reverse engineering is passive; it does not change the software, or result in new software. Reverse engineering efforts produce call graphs and control flow graphs from source code. One type of reverse engineering is redocumentation. Another type is design recovery [Dor02]. Refactoring is program transformation which reorganizes a program without changing its behavior, and is a form of reverse engineering that seeks to improve program structure. (Fow99)

Finally, data reverse engineering has gained in importance over the last few years where logical schemas are recovered from physical databases. (Hen01)

RECOMMENDED REFERENCES FOR SOFTWARE MAINTENANCE

- [Abr93] A. Abran and H. Nguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice*, vol. 5, iss. 2, 1993, pp. 63-90.
- [Arn93] R.S. Arnold, *Software Reengineering*: IEEE Computer Society Press, 1993.
- [Art98] L.J. Arthur, *Software Evolution: The Software Maintenance Challenge*, John Wiley & Sons, 1988.
- [Ben00] K.H. Bennett, "Software Maintenance: A Tutorial," in *Software Engineering*, M. Dorfman and R. Thayer, eds., IEEE Computer Society Press, 2000.
- [Boe81] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.
- [Car90] D.N. Card and R.L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990.
- [Dek92] S. Dekleva, "Delphi Study of Software Maintenance Problems," presented at the International Conference on Software Maintenance, 1992.
- [Dor02] M. Dorfman and R.H. Thayer, eds., *Software Engineering (Vol. 1 & Vol. 2)*, IEEE Computer Society Press, 2002.
- [Gra87] R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, 1987.
- [Hen01] J. Henrard and J.-L. Hainaut, "Data Dependency Elicitation in Database Reverse Engineering," *Proc. of the 5th European Conference on Software Maintenance and Reengineering (CSMR 2001)*, IEEE Computer Society Press, 2001.
- [IEEE610.12-90] IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.
- [IEEE1061-98] IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.
- [IEEE1219-98] IEEE Std 1219-1998, *IEEE Standard for Software Maintenance*, IEEE, 1998.
- [IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.
- [ISO9126-01] ISO/IEC 9126-1:2001, *Software Engineering-Product Quality-Part 1: Quality Model*, ISO and IEC, 2001.
- [ISO14764-99] ISO/IEC 14764-1999, *Software Engineering-Software Maintenance*, ISO and IEC, 1999.
- [ITI01] IT Infrastructure Library, "Service Delivery and Service Support," Stationary Office, Office of Government of Commerce, 2001.
- [Jon98] T.C. Jones, *Estimating Software Costs*, McGraw-Hill, 1998.
- [Leh97] M.M. Lehman, "Laws of Software Evolution Revisited," presented at EWSPT96, 1997.
- [Lie78] B. Lienz, E.B. Swanson, and G.E. Tompkins, "Characteristics of Applications Software Maintenance," *Communications of the ACM*, vol. 21, 1978.
- [Par86] G. Parikh, *Handbook of Software Maintenance*, John Wiley & Sons, 1986.
- [Pfl01] S.L. Pfleeger, *Software Engineering: Theory and Practice*, second ed., Prentice Hall, 2001.
- [Pig97] T.M. Pigoski, *Practical Software Maintenance: Best Practices for Managing your Software Investment*, first ed., John Wiley & Sons, 1997.
- [Pre04] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, sixth ed., McGraw-Hill, 2004.
- [SEI01] Software Engineering Institute, "Capability Maturity Model Integration, v1.1," CMU/SEI-2002-TR-002, ESC-TR-2002-002, December 2001.
- [Sta94] G.E. Stark, L.C. Kern, and C.V. Vowell, "A Software Metric Set for Program Maintenance Management," *Journal of Systems and Software*, vol. 24, iss. 3, March 1994.
- [Tak97] A. Takang and P. Grubb, *Software Maintenance Concepts and Practice*, International Thomson Computer Press, 1997.

APPENDIX A. LIST OF FURTHER READINGS

- (Abr93) A. Abran, "Maintenance Productivity & Quality Studies: Industry Feedback on Benchmarking," presented at the Software Maintenance Conference (ICSM93), 1993.
- (Apr00) A. April and D. Al-Shurougi, "Software Product Measurement for Supplier Evaluation," presented at FESMA2000, 2000.
- (Apr01) A. April, J. Bouman, A. Abran, and D. Al-Shurougi, "Software Maintenance in a Service Level Agreement: Controlling the Customer's Expectations," presented at European Software Measurement Conference, 2001.
- (Apr03) A. April, A. Abran, and R. Dumke, "Software Maintenance Capability Maturity Model (SM-CMM): Process Performance Measurement," presented at 13th International Workshop on Software Measurement (IWSM 2003), 2003.
- (Bas85) V.R. Basili, "Quantitative Evaluation of Software Methodology," presented at First Pan-Pacific Computer Conference, 1985.
- (Bel72) L. Belady and M.M. Lehman, "An Introduction to Growth Dynamics," *Statistical Computer Performance Evaluation*, W. Freiberger, ed., Academic Press, 1972.
- (Ben00) K.H. Bennett and V.T. Rajlich, "Software Maintenance and Evolution: A Roadmap," *The Future of Software Engineering*, A. Finklestein, ed., ACM Press, 2000.
- (Bol95) C. Boldyreff, E. Burd, R. Hather, R. Mortimer, M. Munro, and E. Younger, "The AMES Approach to Application Understanding: A Case Study," presented at the International Conference on Software Maintenance, 1995.
- (Boo94) G. Booch and D. Bryan, *Software Engineering with Ada*, third ed., Benjamin/Cummings, 1994.
- (Cap94) M.A. Capretz and M. Munro, "Software Configuration Management Issues in the Maintenance of Existing Systems," *Journal of Software Maintenance: Research and Practice*, vol. 6, iss. 2, 1994.
- (Car92) J. Cardow, "You Can't Teach Software Maintenance!" presented at the Sixth Annual Meeting and Conference of the Software Management Association, 1992.
- (Car94) D. Carey, "Executive Round-Table on Business Issues in Outsourcing - Making the Decision," *CIO Canada*, June/July 1994.
- (Dor97) M. Dorfman and R.H. Thayer, eds., "Software Engineering," IEEE Computer Society Press, 1997.
- (Dor02) M. Dorfman and R.H. Thayer, eds., *Software Engineering (Vol. 1 & Vol. 2)*, IEEE Computer Society Press, 2002.
- (Fow99) M. Fowler et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- (Gra87) R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, 1987.
- (Gra92) R.B. Grady, *Practical Software Metrics for Project Management and Process Management*, Prentice Hall, 1992.
- (Jon91) C. Jones, *Applied Software Measurement*, McGraw-Hill, 1991.
- (Kaj01) M. Kajko-Mattson, "Motivating the Corrective Maintenance Maturity Model (Cm3)," presented at Seventh International Conference on Engineering of Complex Systems, 2001.
- (Kaj01a) M. Kajko-Mattson, S. Forssander, and U. Olsson, "Corrective Maintenance Maturity Model: Maintainer's Education and Training," presented at International Conference on Software Engineering, 2001.
- (Kho95) T.M. Khoshgoftaar, R.M. Szabo, and J.M. Voas, "Detecting Program Module with Low Testability," presented at the International Conference on Software Maintenance-1995, 1995.
- (Lag96) B. Laguë and A. April, "Mapping for the ISO9126 Maintainability Internal Metrics to an Industrial Research Tool," presented at SESS, 1996.
- (Leh85) M.M. Lehman and L.A. Belady, *Program Evolution: Processes of Software Change*, Academic Press, 1985.
- (Leh97) M.M. Lehman, "Laws of Software Evolution Revisited," presented at EWSPT96, 1997.
- (Lie81) B.P. Lientz and E.B. Swanson, "Problems in Application Software Maintenance," *Communications of the ACM*, vol. 24, iss. 11, 1981, pp. 763-769.
- (McC02) B. McCracken, "Taking Control of IT Performance," presented at InfoServer LLC, 2002.
- (Nie02) F. Niessink, V. Clerk, and H. v. Vliet, "The IT Capability Maturity Model," release L2+3-0.3 draft, 2002, available at <http://www.itservicecmm.org/doc/itscmm-123-0.3.pdf>.
- (Oma91) P.W. Oman, J. Hagemeister, and D. Ash, "A Definition and Taxonomy for Software Maintainability," University of Idaho, Software Engineering Test Lab Technical Report 91-08 TR, November 1991.
- (Oma92) P. Oman and J. Hagemeister, "Metrics for Assessing Software System Maintainability," presented at the International Conference on Software Maintenance '92, 1992.
- (Pig93) T.M. Pigoski, "Maintainable Software: Why You Want It and How to Get It," presented at the Third

Software Engineering Research Forum - November 1993, University of West Florida Press, 1993.

(Pig94) T.M. Pigoski, "Software Maintenance," *Encyclopedia of Software Engineering*, John Wiley & Sons, 1994.

(Pol03) M. Polo, M. Piattini, and F. Ruiz, eds., "Advances in Software Maintenance Management: Technologies and Solutions," Idea Group Publishing, 2003.

(Poo00) C. Poole and W. Huisman, "Using Extreme Programming in a Maintenance Environment," *IEEE Software*, vol. 18, iss. 6, November/December 2001, pp. 42-50.

(Put97) L.H. Putman and W. Myers, "Industrial Strength Software - Effective Management Using Measurement," 1997.

(Sch99) S.R. Schach, *Classical and Object-Oriented Software Engineering with UML and C++*, McGraw-Hill, 1999.

(Sch97) S.L. Schneberger, *Client/Server Software Maintenance*, McGraw-Hill, 1997.

(Sch87) N.F. Schneidewind, "The State of Software Maintenance," *Proceedings of the IEEE*, 1987.

(Som96) I. Sommerville, *Software Engineering*, fifth ed., Addison-Wesley, 1996.

(Val94) J.D. Vallett, S.E. Condon, L. Briand, Y.M. Kim, and V.R. Basili, "Building on Experience Factory for Maintenance," presented at the Software Engineering Workshop, Software Engineering Laboratory, 1994.

APPENDIX A. LIST OF STANDARDS

(IEEE610.12-90) IEEE Std 610.12-1990 (R2002), *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

(IEEE1061-98) IEEE Std 1061-1998, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE, 1998.

(IEEE1219-98) IEEE Std 1219-1998, *IEEE Standard for Software Maintenance*, IEEE, 1998.

(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std. ISO/IEC 12207:95, Standard for Information Technology - Software Life Cycle Processes*, IEEE, 1996.

(IEEE14143.1-00) IEEE Std 14143.1-2000//ISO/IEC14143-1:1998, *Information Technology - Software Measurement-Functional Size Measurement - Part 1: Definitions of Concepts*, IEEE, 2000.

(ISO9126-01) ISO/IEC 9126-1:2001, *Software Engineering-Product Quality - Part 1: Quality Model*, ISO and IEC, 2001.

(ISO14764-99) ISO/IEC 14764-1999, *Software Engineering - Software Maintenance*, ISO and IEC, 1999.

(ISO15271-98) ISO/IEC TR 15271:1998, *Information Technology - Guide for ISO/IEC 12207, (Software Life Cycle Process)*, ISO and IEC, 1998. [Abr93]