CHAPTER 6

SOFTWARE MAINTENANCE

Thomas M. Pigoski
Technical Software Services (TECHSOFT), Inc.
31 West Garden Street, Suite 100
Pensacola, Florida 32501 USA
+1 850 469 0086
tmpigoski@techsoft.com

Table of Contents

Acronyms

CASE	Computer Aided Software Engineering
CM	Configuration Management
CMM	Capability Maturity Model
ICSM	International Conference on Software Maintenance
PSM	Practical Software and Systems Measurement
SCM	Software Configuration Management
SW-CMM	Capability Maturity Model for Software
SQA	Software Quality Assurance
V&V	Verification and Validation
WCRE	Working Conference on Reverse Engineering

1. INTRODUCTION

Software engineering is the application of engineering to software. The life cycle paradigm for software includes: requirements, design, construction, testing, and maintenance. This chapter addresses the maintenance portion of software engineering and the software life cycle.

Software maintenance is an integral part of a software life cycle. However, it has not historically received the same degree of attention as the other phases. Historically, development has had a much higher profile than maintenance in most organizations. This is now changing as organizations strive to obtain the most out of their development investment by keeping software operating as long as possible. Concerns about the Year 2000 (Y2K) rollover did bring significant attention to this important phase. Further, the Open Source paradigm has brought attention to the issue of maintaining code developed by others. Maintenance is also expensive. For these reasons, there is an opportunity to pursue further research to enhance productivity of maintenance activities.

This chapter presents an overview of the Knowledge Area of software maintenance. Brief descriptions of the topics are provided so that the reader can select the appropriate reference material according to his/her needs.

2. DEFINITION OF THE SOFTWARE MAINTENANCE KNOWLEDGE AREA

This section provides a definition of the Software Maintenance Knowledge Area.

Software development efforts result in delivery of a software product that satisfies user requirements. Accordingly, the software product must change or evolve. Once in operation, anomalies are uncovered, operating environments change, and new user requirements surface.

The maintenance phase of the life cycle commences upon delivery but maintenance activities occur much earlier.

Software maintenance sustains the software product throughout its life cycle. Modification requests are logged and tracked, the impact of proposed changes is determined, code is modified, testing is conducted, and a new version of the software product is released. Training is provided to users.

3. BREAKDOWN OF TOPICS FOR THE SOFTWARE MAINTENANCE KNOWLEDGE AREA

The breakdown of topics for software maintenance is a decomposition of software engineering topics that are "generally accepted" in the software maintenance community. They are general in nature and are not tied to any particular domain, model, or business needs. The presented topics can be used by small and medium sized organizations, as well as by larger ones. Organizations should use those topics that are appropriate for their unique situations. The topics are consistent with what is found in current software engineering literature and standards. The common themes of quality, measurement, and standards are included in the breakdown of topics.

The breakdown of topics, along with a brief description of each, is provided in this section. Key references are provided.

3.1. Basic Concepts

3.1.1 Definitions and Terminology [IEEE1219:s3.1.12; ISO12207:s3.1,s5.5; ISO14764:s6.1]

Software maintenance is defined in the IEEE Standard for Software Maintenance, IEEE 1219 [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 annex of the standard.

The ISO/IEC 12207 Standard for Life Cycle Processes [ISO/IEC 12207], 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 existing software product while preserving its integrity." [ISO/IEC 12207] Of note is that ISO/IEC 12207 describes an activity called "Process Implementation." That activity establishes the maintenance plan and procedures that are later used during the maintenance process.

ISO/IEC 14764 [ISO14764], the International Standard for Software Maintenance, defines software maintenance in the same terms as ISO/IEC 12207 and places emphasis on the predelivery aspects of maintenance, e.g., planning.

The SWEBOK definition, generally accepted by software researchers and practitioners, is as follows:

SOFTWARE MAINTENANCE: The totality of activities required to provide cost-effective support to a software system. Activities are performed during the predelivery stage as well as the postdelivery stage. Predelivery activities include planning for postdelivery operations, supportability, and logistics determination. Postdelivery activities include software modification, training, and operating a help desk.

A maintainer is defined by ISO/IEC 12207 as an organization that performs maintenance activities [ISO12207].

ISO/IEC 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 a later section. They are further defined by the tasks in ISO/IEC 12207.

3.1.2 Majority of Maintenance Costs [AH93:pp63-90; Pre97:c27s27.1.2; Pig97:c3]

A common perception of maintenance is that it is merely fixing bugs. However, studies and surveys over the years have indicated that the majority, over 80%, of the maintenance effort is used for non-corrective actions [AH 93] [Pre97] [Pig97]. This perception is perpetuated by users submitting problem reports that in reality are major enhancements to the system. This inclusion of enhancement requests with problem reports contributes to some of the misconceptions regarding maintenance. Software evolves over its life cycle, as evidenced by the fact that over 80% of the effort after initial delivery goes to implement non-corrective actions. Thus, maintenance is similar to software development, although some unique processes are employed.

The focus of software development is to solve problems or to obtain business advantage through producing code. The generated code implements stated requirements and should operate correctly. Maintainers look back at development products and also the present by working with users and operators. Maintainers also look forward to anticipate problems and to consider functional changes.

3.1.3 The Nature of Maintenance [Pfl98:c10s10.2]

Pfleeger [Pfl98] states that maintenance has a broader scope than development, with more changes to track and control. Thus, configuration management is an important aspect of software evolution and maintenance.

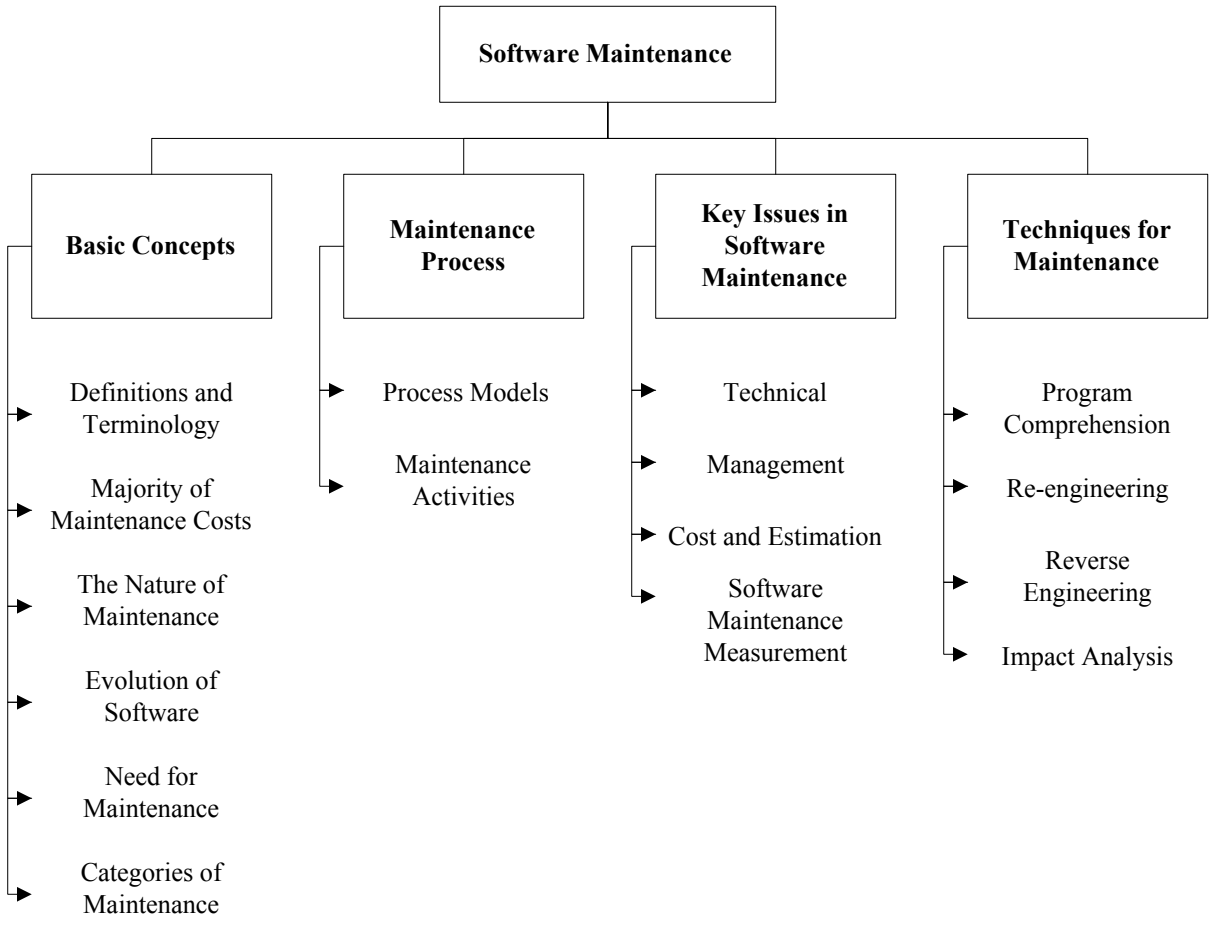

Figure 1 Summary of the Software Maintenance Breakdown

Maintenance, however, can learn from the development effort. Contact with the developers and early involvement by the maintainer helps the maintenance effort. However, it is difficult sometimes when the developers are no longer around. Maintenance must take the products of the development, e.g., code, documentation, and evolve/maintain them over the life cycle. Chapter 10 of the Guide to the SWEBOK discusses how tools can aid maintenance.

3.1.4 Evolution of Software [Leh97:pp108-124; Pfl98: c10s10.1;Art88:c1s1.0,s1.1,s1.2,c11,s1.1,s1.2]

The area of software maintenance and evolution of systems was first addressed by Lehman in 1969. His research led to an investigation of the evolution of OS/360 [LB85] and continues today on the Feedback, Evolution, and Software Technology (FEAST) research at Imperial College, England.

Over a period of twenty years, that research led to the formulation of eight Laws of Evolution [Leh97]. Simply put, Lehman stated that maintenance is really evolutionary developments and that maintenance decisions are aided by understanding what happens to systems (and software) over time. Others state that maintenance is really continued development, except that there is an extra input (or constraint) – the existing software system.

Key points from Lehman include that large systems are never complete and continue to evolve. As they evolve, they grow more complex unless some action is taken to reduce the complexity. As systems demonstrate regular behavior and trends, these can be measured and predicted. Pfleeger [Pfl98] and Arthur [Art88] have excellent discussions regarding software evolution.

3.1.5 Need for Maintenance [Pfl98:c10.s10.2; Pig97: c2s2.3; TG97:c1]

Maintenance is needed to ensure that the system continues to satisfy user requirements. Maintenance is applicable to systems developed using any software development model (e.g., spiral). The system changes due to corrective and non-corrective software actions. Maintenance must be performed in order to:

Correct errors.

Correct requirements and design flaws.

Improve the design.

Make enhancements.

Interface with other systems.

Convert programs so that different hardware, software, system features, and telecommunications facilities can be used.

Migrate legacy systems.

Retire systems.

The four major aspects that maintenance focuses on are [Pfl98]:

Maintaining control over the system's day-to-day functions.

Maintaining control over system modification.

Perfecting existing acceptable functions.

Preventing system performance from degrading to unacceptable levels.

Accordingly, software must evolve and be maintained.

3.1.6 Categories of Maintenance [Art88:c1s1.2; DT97:c8s5; IEEE1219:s3.1.1,s3.1.2,s3.1.7,A.1.7; ISO14764:s4.1,s4.3, s4.10,s4.11,s6.2; Pfl98: c10s10.2; Pig97:c2s2.3]

Lehman developed the concept of software evolution. E. B. Swanson of UCLA was one of the first to examine what really happens in evolution and maintenance, using empirical data from industry maintainers. Swanson believed that, by studying the maintenance phase of the life cycle, a better understanding of the maintenance phase would result. Swanson was able to create three different categories of maintenance: corrective, adaptive, and perfective. [Art88] [DT97]. There have been updated and a new category has been defined by the International Organization of Standards (IS0) in the Standard for Software Maintenance standard ISO/IEC 14764, [ISO14764] and by the IEEE Computer Society [IEEE 1219]**.** The categories of maintenance defined by ISO/IEC are 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.

The ISO Standard on Software Maintenance [ISO14764] classifies Adaptive and Perfective maintenance as enhancements. It also classifies Corrective and Preventive maintenance as corrections. Preventive maintenance, the newest category, is defined as maintenance performed for the purpose of preventing problems before they occur. Preventive maintenance is most often performed on software products where safety is critical.

3.2. Maintenance Process

The need for software processes is well documented. The Capability Maturity Model for Software (SW-CMM) provides a means to measure levels of maturity. Of importance, is that there is a direct correlation between levels of maturity and cost savings. The higher the level of maturity, the greater the cost savings. The SW-CMM applies equally to maintenance and maintainers should have a documented maintenance process

3.2.1 Maintenance Process Models [IEEE1219:s4; ISO14764:s8; ISO12207:s5.5; Pig97:c5; TG97:c2; Par86:c7s1]

Process models provide needed operations and detailed inputs/outputs to those operations. Maintenance process models are provided in the software maintenance standards, IEEE 1219 [IEEE 1219] and ISO/IEC 14764 [ISO14764].

The maintenance process model described in IEEE 1219 [IEEE 1219], the Standard for Software Maintenance, starts the software maintenance effort during the post-delivery stage and discusses items such as planning for maintenance and measures outside the process model. That process model with the IEEE maintenance phases is depicted in Figure 2.

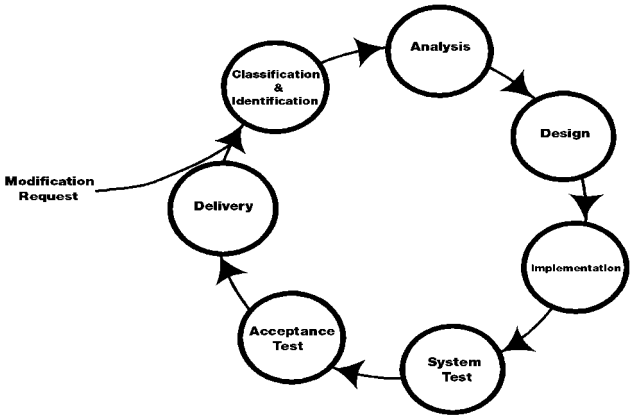

Figure 2 The IEEE Maintenance Process Activities

ISO/IEC 14764 [ISO14764] is an elaboration of the maintenance process of ISO/IEC 12207 [ISO12207]. The activities of the ISO/IEC maintenance process are similar to those of IEEE although they are aggregated a little differently. The maintenance process activities developed

by ISO/IEC are shown in Figure 3.

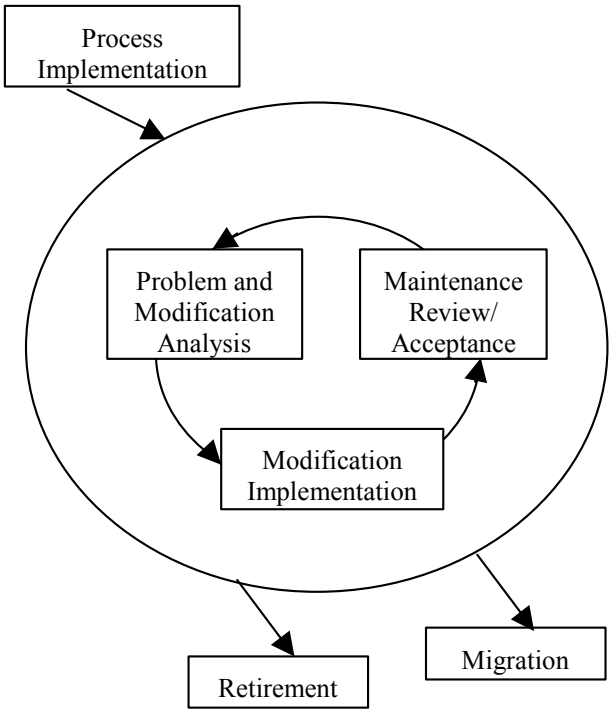

Figure 3 ISO/IEC Maintenance Process Activities

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

Process Implementation tasks are:

Develop maintenance plans and procedures.

Establish procedures for Modification Requests.

Implement the CM process.

Problem and Modification tasks are:

Perform initial analysis.

Verify the problem.

Develop options for implementing the modification.

Document the results.

Obtain approval for modification option.

Modification Implementation tasks are:

Perform detailed analysis.

Develop, code, and test the modification.

Maintenance Review/Acceptance tasks are:

Conduct reviews.

Obtain approval for modification.

Migration tasks are:

Ensure that migration is in accordance with ISO/IEC 12207.

Develop a migration plan.

Notify users of migration plans.

Conduct parallel operations.

Notify user that migration has started.

Conduct a post-operation review.

Ensure that old data is accessible.

Software Retirement tasks are:

Develop a retirement plan.

Notify users of retirement plans.

Conduct parallel operations.

Notify user that retirement has started.

Ensure that old data is accessible.

Takang and Grubb [TG97] provide a history of maintenance process models leading up to the development of the IEEE and ISO/IEC process models. A good overview of a generic maintenance process is given by Parikh [Par86]

3.2.2 Maintenance Activities

Maintenance activities are similar to those of software development. Maintainers perform analysis, design, coding, testing, and documenting. Maintainers must track requirements just as they do in development. Maintainers must update documentation as baselines change. However, for software maintenance, the activities involve processes unique to maintenance. Chapter 10 discusses how tools can be used to help in the maintenance effort.

3.2.2.1 Unique Activities [Pfl98:c10s10.2; Art88:c3; DT97: c8s9.1; IEEE1219:s4.1,s4.2; ISO14764: s8.2.2.1, s,8.3.2.1]

Maintainers must possess an intimate knowledge of the code's structure and content [Pfl98]. That knowledge is used by maintainers to perform impact analysis. Impact analysis identifies all systems and system products affected by a 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 and often called a problem report, must first be analyzed and translated into software terms [DT97]. The maintainer then identifies the affected components. Several potential solutions are provided and then a recommendation is made as to the best course of action.

Problem solving skills are very important for maintenance. Maintainers must also be concerned about the "ripple effect" of any proposed changes.

3.2.2.2 Supporting Activities [IEEE1219:A.7,A.11; Pig97: c10s10.2,c18; ISO12207:c6,c7]

Maintainers may also perform supporting activities such as configuration management (CM), verification and validation, quality assurance, reviews, audits, and conducting user training. Often these supporting activities are performed by separate entities. The IEEE Standard for

Software Maintenance, IEEE 1219 [IEEE 1219], describes CM as a critical element of the maintenance process. CM procedures should provide for the verification, validation, and certification of each step required to identify, authorize, implement, and release the software product. Training of maintainers, a supporting process, is also a needed activity [Pig97] [ISO12207].

3.2.2.2.1 Configuration management [ISO12207:s6.2; IEEE1219: A.11; Art88:c2,c10; Pfl98:c10s10.5; TG97:c7]

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. SCM provides support and makes the job of the maintainer easier. Chapter 7 of the Guide to the SWEBOK provides details of SCM and discusses the process by which change requests are submitted, evaluated, and approved. SCM for maintenance is different than for development in that a change request initiates the maintenance process. The SCM process is implemented by developing and following a CM Plan and operating procedures. Maintainers participate in Configuration Control Boards to determine when enhancements should stop and perhaps migration is necessary. Problem severity is often used to decide how and when a problem will be fixed.

3.2.2.2.2 Quality [ISO12207:s6.3; IEEE1219:A.7; Art98: c7s4]

It is not sufficient 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) and V&V must be selected in concert with all other processes to achieve the level of quality desired. This is implemented by developing and following SQA and V&V plans and procedures. Details of software quality are covered in chapter 11 of the Guide to the SWEBOK.

3.2.2.2.3 Maintenance Planning Activity [IEEE1219:A.3; ISO14764:s7; Pig97:c7,c8]

An important activity for software maintenance is planning. Whereas developments typically can last for 1-2 years, the operation and maintenance phase typically lasts for many years. Developing accurate estimates of resources is a key element of maintenance planning. Those resources, which include costs, should be included in project planning budgets. Maintenance planning should begin with the decision to develop a new system and should consider quality objectives. A concept and then a maintenance plan should be developed. The concept for maintenance should address:

The scope of software maintenance.

The tailoring of the postdelivery process.

The designation of who will provide maintenance.

An estimate of life cycle costs.

Once the maintenance concept is determined, the next step is to develop the maintenance plan. The maintenance plan should be prepared during software development and should specify how users will request modifications or report problems. Maintenance planning [Pig97] is addressed in IEEE 1219 [IEEE 1219]and ISO/IEC 14764. [ISO14764] ISO/IEC14764 [ISO14764] provides guidelines for a maintenance plan.

3.3. Key Issues in Software Maintenance

It is important to understand that software maintenance provides unique technical and management problems for software engineers. Trying to find a defect in a 500K line of code system that the maintainer did not develop is a challenge for the maintainer. 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, is also a challenge. The following discusses some of the technical and management problems relating to software evolution and maintenance.

3.3.1 Technical Problems

3.3.1.1 Limited understanding [Pfl98:c10s10.3; TG97:c3; DT97: c8s11.4]

Practitioners and researchers indicate that some 40% to 60% of the maintenance effort is devoted to understanding the software to be modified. Thus, the topic of program comprehension is one of interest to maintainers. Comprehension is more difficult for text-based representation. It is often difficult to trace the evolution of the software through its versions, changes are not documented, and the developers are usually not around to explain the code. Thus, maintainers have a limited understanding of the software and must learn the software on their own.

3.3.1.2 Testing [Pfl98:c10s10.3; Art88:c9]

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 system or component to verify the modifications have not caused unintended effects, is important to maintenance. Research efforts into areas such as “slicing” look at this topic. Finding time to test is often difficult [Plf98]. Chapter 5 of the Guide to the SWEBOK provides details of testing.

3.3.1.3 Impact analysis [DT97:c8s10.1-3; Pfl98: c10s10.5; Art88:c3]

The software and the organization must both undergo impact analysis. Critical skills, documentation, and processes are needed for this area. Impact analysis is necessary for risk abatement. Software designed for maintainability facilitates impact analysis.

3.3.1.4 Maintainability [ISO14764:s6.8s6.8.1;Pfl98: c8s8.4;Pig97:c16]

The IEEE Computer Society [IEEE610.12] 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. Maintainability features must be incorporated into the software development effort to reduce life cycle costs. If this is done, the quality of evolution and maintenance of the code can improve. Maintainability is often a problem in maintenance because maintainability is not incorporated into the software development process, documentation is lacking, and program comprehension is difficult. Maintainability can be achieved by including it in requirements, design, and construction. Chapters 2, 3, and 4 provide details of these topics. Maintainability can be enhanced by defining coding standards, documentation standards, and standard test tools in the software development phase of the life cycle.

3.3.2 Management

3.3.2.1 Alignment with organizational issues [DT97: c8s6; Pfl98:c10s10.3]

Dorfman and Thayer [DT97] relate that return on investment is not clear with maintenance. Thus, there is a constant struggle to obtain resources.

3.3.2.2 Staffing [Pfl98:c10s10.3; Dek92:pp10-17; Par86: c4s8-s11; DT97:c8s6]

Maintenance personnel often are viewed as second class citizens [Pfl98] and morale suffers [DT97]. Maintenance is not viewed as glamorous work. Deklava provides a list of staffing related problems based on survey data [Dek92].

3.3.2.3 Process issues [DT97:c8s3]

Maintenance requires several activities that are not found in software development, (e.g., help desk support). These present challenges to management [DT97].

3.3.2.4 Organizational Aspects of Maintenance

The team that develops the software is not always used to maintain the system once it is operational. A maintainer must be identified and there are several options as discussed below.

3.3.2.4.1 *The Maintainer* [Pfl98:c10s10.2; Pig97:c2s2.5; Par86: c4s7; TG97:c8]

Often, a separate team (or maintainer) is employed to ensure that the system runs properly and evolves to satisfy changing needs of the users. There are many pros and cons to having the original developer or a separate team maintain the software [Pfl98] [Pig97] [Par86]. That decision should be made on a case-by-case basis.

3.3.2.4.2 *Outsourcing* [DT97:c8s7;Pig97: c9s9.1,s9.2]

Outsourcing of maintenance is becoming a major industry. Large corporations are outsourcing entire operations, including software maintenance. More often outsourcing is done for peripheral software, as companies are unwilling to release the software used in its core business. One of the major challenges is for the outsource maintenance company to determine the scope of the effort. Outsourcing companies typically spend a number of months assessing the software before it will accept a contract [DT97]. Another challenge is the transition of the software to the outsourced company [Pig97].

3.3.2.4.3 *Organizational Structure* [Pig97:c12s12.1-s12.3]

Based on the fact there are almost as many organizational structures as there are software maintenance organizations, an organizational structure for maintenance is best developed on a case-by-case basis. What is important is the delegation or designation of maintenance responsibility to a group [Pig97], regardless of the organizational structure. As with other efforts, maintenance will only be successful with full management support.

3.3.3 Maintenance Cost and Maintenance Cost Estimation

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

3.3.3.1 Cost [Pfl98:c10s10.3; Art88:c3; Pig97:c3s3.1-3; Pre97: c27s27.2.2]

Maintenance consumes a major share of life cycle costs. Understanding the categories of maintenance helps to understand why maintenance is so costly. Also understanding the factors that influence the maintainability of a system can help to contain costs. Pfleeger [Pfl98] addresses some of the technical and non-technical factors affecting maintenance.

Impact analysis identifies all systems and system products affected by a change request and develops an estimate of the resources needed to accomplish the change [Art88]. It is performed after a change request enters the CM process. It is used in concert with the cost estimation techniques discussed below.

3.3.3.2 Cost estimation [Boe81:c30; Jon98:c27; Pig97:c8; Pfl98:c10s10.3]

Maintenance cost estimates are affected by many technical and non-technical factors. Primary approaches to cost

estimating include use of parametric models and experience. Most often a combination of these is used to estimate costs.

3.3.3.3 Parametric models [Boe81:c30; Jon98:c27; Pfl98:c10s10.3]

One of the works in the area of parametric models for estimating was performed by Boehm [Boe81]. COCOMO (derived from COnstructive COst Model), puts the software life cycle and the quantitative life cycle relationships into a hierarchy of software cost-estimation models [Pfl98]. Of significance is that data from past projects is needed in order to use the models. Jones [Jon98] discusses all aspects of estimating costs including function points, and provides a detailed chapter on maintenance estimating. Chapter 8 of the Guide to the SWEBOK provides additional details regarding models.

3.3.3.4 Experience *[Pig97:c8; ISO14764:s7,s7.2,s7.2.1, c7s7.2.4]*

Experience should be used to augment data from parametric models. Sound judgment, reason, a work breakdown structure, educated guesses, and use of empirical/historical data are several approaches. Clearly the best approach to maintenance estimation is to use empirical data and experience. That data should be provided as a result of a measurement program. In practice, cost estimation relies much more on experience than parametric models. The Software Engineering Institute has conducted research into performing cost estimation based on historical data.

3.3.4 Software Maintenance Measurement [GC87:c2; TG97: c6s6.1-3; AI98:A.2]

Software life cycle costs are growing and a strategy for maintenance is needed. Software measurement need to be a part of that strategy. Grady and Caswell [GC87] discuss establishing a corporate-wide software measures program. The Practical Software and Systems Measurement (PSM) project describes an issue-driven measurement process [http://www.psmsc.com] that is used by many organizations and is quite practical. Software measures are vital for software process improvement but the process must be measurable. Additional discussion of measurement is contained in chapters 8 and 11 of the Guide to the SWEBOK.

3.3.4.1 Specific Measure*s [CG90:s2-3; SKV94:pp239-249; IEEE1219:Table3; Pig97:c14s14.6; TG97: c6s6.4]*

There are software measures that are common to all efforts and the Software Engineering Institute (SEI) identified these as: size; effort; schedule; and quality [Pig97]. Those are a good starting point for a maintainer.

Takang and Grubb [TG97] group software measures into areas of: size; complexity; quality; understandability; maintainability; and cost estimation.

Documentation regarding specific software measures to use in maintenance is not often published. Typically generic software engineering measures are used and the maintainer determines which ones are appropriate for their organization. IEEE 1219 [IEEE 1219] provides suggested measures for software programs. Stark, et al [SKV94] provide a suggested list of software maintenance measures used at NASA's Mission Operations Directorate. That list includes:

Software size

Software staffing

Maintenance request number/status

Software enhancement numbers/status

Computer resource utilization

Fault density

Software volatility

Discrepancy report open duration

Break/fix ratio

Software reliability

Design complexity

Fault type distribution

3.4. Techniques for Maintenance

Effective software maintenance is performed using techniques specific to maintenance. The following provides some of the best practice techniques used by maintainers.

3.4.1 *Program Comprehension* [Arn92:c14; DT97: c8s11.4; TG97:c3]

Programmers spend considerable time in reading and comprehending programs in order to implement changes. Code browsers are a key tool in program comprehension. Clear and concise documentation can aid in program comprehension. Based on the importance of this subtopic, an annual IEEE Computer Society workshop is now held to address program comprehension. The website http://www.seg.iit.nrc.ca/projects/easse provides a number of papers on comprehension and tools for assisting comprehension processes. Takang and Grubb [TG97] provide a detailed chapter on comprehension.

3.4.2 *Re-engineering* [Arn92:c1,c3-6, c8s11.4; IEEE1219: B.2; DT97:c8s11.4]

Re-engineering is defined as the examination and alteration of the subject system to reconstitute it in a new form, and the subsequent implementation of the new form. Dorfman and Thayer [DT97] state that re-engineering is the most radical (and expensive) form of alteration. Others believe that re-engineering can be used for minor changes. Re-engineering is often not undertaken to improve maintainability but is used to replace aging legacy systems.

Arnold [Arn92] provides a comprehensive compendium of topics, e.g., concepts, tools and techniques, case studies, and risks and benefits associated with re-engineering. Refactoring, a program transformation that reorganizes a program without changing its behavior, is now being used in reverse engineering to improve the structure of object-oriented programs.

3.4.3 *Reverse engineering* [Arn92:c12; DT97:c8s11.3; IEEE1219:B.3; TG97:c4]

Reverse engineering is the process of analyzing a subject system to identify the system's components and their inter-relationships and to create representations of the system in another form or at higher levels of abstraction. Reverse engineering is passive, it does not change the system, or result in a new one. A simple reverse engineering effort may merely produce call graphs and control flow graphs from source code. One type of reverse engineering is redocumentation. Another type is design recovery [DT97]. Date Reverse Engineering has gained great importance over the last few years. Reverse engineering topics are discussed at the annual Working Conference on Reverse Engineering (WCRE).

3.4.4 *Impact Analysis* [Plf98:c10s10.5; Art88:c3]

Impact analysis identifies all systems and system products affected by a change request and develops an estimate of the resources needed to accomplish the change [Art88]. It is performed after a change request enters the configuration management process. Arthur [Art88] states that the objectives of impact analysis are:

> Determine the scope of a change in order to plan and implement work.
>
> Develop accurate estimates of resources needed to perform the work.
>
> Analyze the cost/benefits of the requested change.
>
> Communicate to others the complexity of a given change.

Resources

Beside the references listed in this chapter, there are other resources available to learn more about software maintenance. The IEEE Computer Society sponsors the annual *International Conference on Software Maintenance (ICSM)*. That conference, started in 1983, provides a Proceedings, which incorporates numerous research and practical industry papers concerning evolution and maintenance topics. Other venues, which address these topics, include:

4. BREAKDOWN RATIONALE

The breakdown of topics for software maintenance is a decomposition of software engineering topics that are "generally accepted" in the software maintenance community. They are general in nature. There is agreement in the literature and in the standards on the topics.

A detailed discussion of the rationale for the proposed breakdown, keyed to the Guide to the SWEBOK development criteria, is given in Appendix B. The following is a narrative description of the rationale for the breakdown.

The Basic Concepts sub-area was selected as the initial topic in order to introduce Software Maintenance. The subtopics are needed to provide definitions and to emphasize why there is a need for maintenance. Categories are critical to understand the underlying meaning of maintenance.

Maintenance Process is needed to provide the current references and standards needed to implement the maintenance process.

The Maintenance Activities sub-topic is needed to differentiate maintenance from development and to show the relationship to other software engineering activities.

The sub-area on the Key Issues of Software Maintenance was chosen to ensure that the software engineers fully comprehended these problems.

Every organization is concerned with who will perform maintenance. The Management topic provides some options regarding who can perform maintenance. Every software maintenance reference discusses the fact that maintenance consumes a large portion of the life cycle costs. The topic on Cost and Cost Estimation was provided to ensure that the readers select references to help with this difficult task.

The Software Maintenance Measurement topic is one that is not addressed very well in the literature. Most maintenance books barely touch on the topic. Measurement information is most often found in generalized measurement books. This topic was chosen to highlight the need for unique maintenance measures and to provide specific maintenance measurement references.

The Techniques topic was provided to introduce some of the generally accepted techniques used in maintenance operations.

5. MATRIX OF TOPICS VS. REFERENCE MATERIAL

Topics	AH 93	IEEE 610.12	AI 98	Arn 92	Art 88	Boe 81	CG 90	Dek 92	DT 97	GC 87	IEEE 1219	ISO 12207	ISO 14764	Jon 98	Leh 97	Par 86	Pfl 98	Pig 97	Pre 97	SKV 94	TG 97
1. Basic Concepts																					
1.1 Definitions and Terminology											s3.1.12	s3.1, s5.5	s6.1								
1.2 Majority of Maintenance Costs	pp 63-90																	c3	c27 s27.1.2		
1.3 Nature of Maintenance																	c10 s10.2				
1.4 Evolution of Software					c1 s1.0 s1.1 s1.2 c11 s1.1 s1.2										pp 108-124		c10 s10.1				
1.5 Need for Maintenance																	c10 s10.2	c2 s2.3			c1
1.6 Categories of Maintenance					c1 s1.2				c8 s5		c3 s3.1, s3.1.1, s3.1.2, s3.1.7, A.1.7		s4.1 s4.3 s4.10 s4.11 s6.2				c10 s10.2	c2 s2.3			
2. Maintenance Process																					
2.1 Maintenance Process Models											s4	s5.5	s8			c7,s1		c5			c2
2.2 Maintenance Activities																					
Unique Activities					c3				c8 s9.1		s4.1, s4.2		s8.2.2.1, s8.3.2.1				c10 s10.2				
Supporting Activities											A.7, A.11	c6,c7						c10 s10.2, c18			
Configuration Management					c2 c10						A.11	s6.2					c10, s10.5				c7
Quality					c7 s4						A.7	s6.3									
Maintenance Planning Activity											A.33		c7					c7,c8			
3. Key Issues in Software Maintenance																					
3.1 Technical																					
Limited Understanding									c8 s11.4								c10 s10.3				c3
Testing					c9												c1 s10.3				
Impact Analysis					c3				c8 s10.1 s10.2 s10.3								c10 s10.5				
Maintainability		s3											s6.8, s6.8.1				c8 s8.4	c16			
3.2 Management																					
Alignment with organizational issues									c8 s6								c10 s10.3				
Staffing								pp 10-17	c8 s6							c4, s8-11	c10, s10.3				c1, s1.8
Process issues									c8, s3												
Organizational																					
The Maintainer																c4 s7	c10 s10.2	c2 s2.5			c8
Outsourcing									c8 s7									c9 s9.1, s9.2			
Organizational Structure																		c12 s12.1 s12.2 s12.3			

Topics	AH 93	IEEE 610.12	AI 98	Arn 92	Art 88	Boe 81	CG 90	Dek 92	DT 97	GC 87	IEEE 1219	ISO 12207	ISO 14764	Jon 98	Leh 97	Par 86	Pfl 98	Pig 97	Pre 97	SKV 94	TG 97
3.3 Maintenance Cost and Maintenance Cost Estimation																					
Cost					c3												c10 s10.3	c3 s3.1-3	c27 s27.2.2		
Cost estimation						c30								c27			c10 s10.3	c8			
Parametric models						c30								c27			c10 s10.3				
Experience													s7 s7.2, s7.2.1, s7.2.4					c8			
3.4 Software Maintenance Measurement							s2-3				Table 3							c14 s14.6		pp 239-249	c6 s6.4
4. Techniques for Maintenance																					
4.1 Program Comprehension				c14					c8 s11.4												c3
4.2 Re-engineering				c1,3-6					c8 s11.4		B.2										
4.3 Reverse Engineering				c12					c8 s11.3		B.3										c4
4.4 Impact Analysis					c3												c10 s10.5				

6. RECOMMENDED REFERENCES FOR SOFTWARE MAINTENANCE

The following set of references provides a strong foundation to acquire knowledge on specific topics identified in the breakdown. They were chosen to provide coverage of all aspects of software maintenance. Priority was given to standards, maintenance specific publications, and then general software engineering publications.

References

[AH93] A. Abran and H. Nguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice*, Vol 5, no 2, 1993 [pp63-90].

[AI98] ANSI/IEEE STD 1061. *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Computer Society Press, 1998. [s4, A.1, A.2]

[Arn92] R.S. Arnold. *Software Reengineering*. IEEE Computer Society, 1993. [c1,c3-6,c12,c14]

[Art88] L.J. Arthur. *Software Evolution: The Software Maintenance Challenge*. John Wiley & Sons, 1988. [c1s1.0,s1.1,s1.2; c2, c3, c7s4, c9, c10,c11s1.1,s1.2]

[Boe81] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981. [c30]

[CG90] D.N. Card and R. L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990. [s1.1,1.3,c2-3]

[Dek92] S. Dekleva. Delphi Study of Software Maintenance Problems. *Proceedings of the International Conference on Software Maintenance,* 1992. [pp10-17]

[DT97] M. Dorfman and R. H. Thayer. *Software Engineering.* IEEE Computer Society Press, 1997. [c8s3, c8s5, c8s6, c8s7, c8s9.1, c8s10.1-3, c8s11.3-4]

[GC87] R.B. Grady and D. L. Caswell. *Software Metrics: Establishing a Company-wide Program*. Prentice-Hall, 1987. [c2, c3]

[IEEE610.12] IEEE STD 610.12: *IEEE Standard Glossary of Software Engineering Terminology*, 1990. [s3]

[IEEE1219] *IEEE STD 1219: Standard for Software Maintenance*, 1998. [s3.1.1,s3.1.2,s3.1.7,s4,s4.1,s4.2, A.1.7,A.3,A.7,A.11, Table3, B.2-3]

[ISO12207] *ISO/IEC 12207: Information Technology-Software Life Cycle Processes*, 1995. [s3.1, s5.5, c6, s6.2,s6.3, c7]

[ISO14764] *ISO/IEC 14764: Software Engineering-Software Maintenance*, 2000. [s4.1,s4.3,s4.10,s4.11,s6.1, s6.2,s6.8,s6.8.1,s7,s7.2,s7.2.1,s7.2.4,s8,s8.2.2.1,s8.3.2.1]

[Jon98] T. C. Jones. *Estimating Software Costs.* McGraw-Hill, 1998. [c27]

[Leh97] M.M Lehman, Laws of Software Evolution Revisited, EWSPT96, October 1996, LNCS 1149, Springer Verlag, 1997. [pp108-124]

[Par86] G. Parikh. *Handbook of Software Maintenance*. John Wiley & Sons, 1986. [c4s7-11, c7s1]

[Pfl98] S.L. Pfleeger. *Software Engineering—Theory and Practice.* Prentice Hall, 1998. [c8s8.4,c10s10.1,s10.2, s10.3,s10.5]

[Pig97] T.M. Pigoski. *Practical Software Maintenance: Best Practices for Managing your Software Investment.*

Wiley, 1997. [c2s2.3,s2.5, c3, c3s3.1-3, c5, c7, c8, c9s9.1-2, c10s10.2, c12s12.1-3, c14s4-5, c14 s14.6, c16, c18]

[Pre97] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, fourth edition, 1997. [c27s27.2.1-2]

[SKV94] 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, no. 3, March 1994. [pp239-249]

[TG97] A. Takang and P. Grubb. *Software Maintenance Concepts and Practice*. International Thomson Computer Press, 1996. [c1, c1s1.8, c2, c3, c4, c6s6.1-4, c7, c8]

APPENDIX A – LIST OF FURTHER READINGS

Beside the recommended references listed in this chapter, there are other resources available to learn more about software maintenance. The IEEE Computer Society sponsors the annual *International Conference on Software Maintenance (ICSM)*. That conference, started in 1983, provides a Proceedings, which incorporates numerous research and practical industry papers concerning evolution and maintenance topics. Other venues, which address these topics, include:

The Workshop on Software Change and Evolution (SCE). [HTTP://www.dur.ac.uk/~dcs0elb/csm/sce99/]

Manny Lehman's work on the FEAST project at the Imperial College in England continues to provide valuable research into software evolution. [HTTP://www-dse.doc.ic.uk/~mml/]

The International Workshop on Empirical Studies of Software Maintenance (WESS). [HTTP://computer.org/conferences/calendar/htm]

The Research Institute for Software Evolution (RISE) at the University of Durham, England, concentrates its research on software maintenance and evolution. [HTTP://www.dur.ac.uk/csm]

The Seventh Working Conference on Reverse Engineering (WCRE-2000). [HTTP://computer.org/conferences/calendar/htm]

The Conference on Software Maintenance and Reengineering (CSMR). [HTTP://www.uni-koblenz.de/ ~ist/SCSMR2000/]

The *Journal of Software Maintenance*, published by John Wiley & Sons, also is an excellent resource for maintenance.

A list of additional readings is also provided to identify additional reference material for the Knowledge Area of Software Maintenance. These references also contain generally accepted knowledge.

References

[AH93] A. Abran and H. Hguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice*, Vol 5, no 2, 1993.

[AI98] ANSI/IEEE STD 1061. *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Computer Society Press, 1998.

[Arn92] R.S. Arnold. *Software Reengineering*. IEEE Computer Society, 1992.

[Art88] L.J. Arthur. *Software Evolution: The Software Maintenance Challenge*. John Wiley & Sons, 1988.

[Bas85] V.R. Basili, "Quantitative Evaluation of Software Methodology," *Proceedings First Pan-Pacific Computer Conference,* September 1985.

[Boe81] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[BBHMMY] C. Boldyreff, E. Burd, R. Hather, R. Mortimer, M. Munro, and E. Younger, "The AMES Approach to Application Understanding: A Case Study," *Proceedings of the International Conference on Software Maintenance-1995*, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[CM94] M.A. Capretz and M. Munro, "Software Configuration Management Issues in the Maintenance of Existing Systems," *Journal of Software Maintenance,* Vol. 6, no.2, 1994.

[CG90] D.N. Card and R. L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990.

[Car92] J. Cardow, "You Can't Teach Software Maintenance!," *Proceedings of the Sixth Annual Meeting and Conference of the Software Management Association,* 1992.

[Dek92] S. M. Dekleva. Delphi Study of Software Maintenance Problems. *Proceedings of the International Conference on Software Maintenance,* 1992.

[DT97] M. Dorfman and R. H. Thayer. *Software Engineering*. IEEE Computer Society Press, 1997.

[GC87] 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 Improvement*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1992.

[IEEE610.12] IEEE STD 610.2: *IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[IEEE1219] *IEEE STD 1219: Standard for Software Maintenance,* 1998.

[ISO12207] *ISO/IEC 12207: Information Technology-Software Life Cycle Processes, 1995.*

[ISO14764] *ISO/IEC 14764: Software Engineering-Software Maintenance,* 2000.

[ISO15271] ISO/IEC TR 15271, *Information Technology - Guide for ISO/IEC 12207, (Software Life Cycle Process)*

[Jon98] T.C. Jones. *Estimating Software Costs.* McGraw-Hill, 1998.

[LB85] M.M. Lehman and L.A. Belady, *Program Evolution – Processes of Software Change*, Academic Press Inc. (London) Ltd., 1985.

[Leh97] M.M. Lehman, Laws of Software Evolution Revisited, EWSPT96, October 1996, LNCS 1149, Springer Verlag, 1997.

[KSV95] T.M. Khoshgoftaar, R.M. Szabo, and J.M. Voas, "Detecting Program Module with Low Testability," *Proceedings of the International Conference on Software Maintenance-1995*, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[OHA91] 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.

[OH92] P. Oman and J. Hagemeister, "Metrics for Assessing Software System Maintainability," *Proceedings of the International Conference on Software Maintenance-1992,* IEEE Computer Society Press, Los Alamitos, CA, 1992.

[Par86] G. Parikh. *Handbook of Software Maintenance*. John Wiley & Sons, 1986.

[Pfl98] S. L. Pfleeger. *Software Engineering—Theory and Practice.* Prentice Hall, 1998.

[Pig93] T.M. Pigoski, "Maintainable Software: Why You Want It and How to Get It," *Proceedings of the Third Software Engineering Research Forum-November 1993*, University of West Florida Press, Pensacola, FL, 1993.

[Pig94] T.M. Pigoski. "Software Maintenance," *Encyclopedia of Software Engineering,* John Wiley & Sons, New York, NY, 1994.

[Pig97] T.M. Pigoski. *Practical Software Maintenance: Best Practices for Managing your Software Investment.* Wiley, 1997.

[PM97] L.H. Putman and W. Myers. *Industrial Strength Software – Effective Management Using Measurement*, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[Pre97] R.S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, fourth edition, 1997.

[Scha99] S.R. Schach, *Classical and Object-Oriented Software Engineering With UML and C++,* McGraw-Hill, 1999

[Sch87] N.F. Schneidewind. *The State of Software Maintenance*. *Proceedings of the IEEE*, 1987.

[Schn97] S.L. Schneberger, *Client/Server Software Maintenance,* McGraw-Hill, 1997.

[Som01] I. Sommerville. *Software Engineering.* Addison-Wesley, sixth edition, 2001.

[SKV94] G.E. Stark, L. C. Kern, and C. V. Vowell. *A Software Metric Set for Program Maintenance Management*. Journal of Systems and Software, 1994.

[TG97] A. Takang and P. Grubb. *Software Maintenance Concepts and Practice.* International Thomson Computer Press, 1997.

[VCBKB] J.D. Vallett, S.E. Condon, L. Briand, Y.M. Kim and V.R. Basili, "Building on Experience Factory for Maintenance," *Proceedings of the Software Engineering Workshop,* Software Engineering Laboratory, 1994.

APPENDIX B – REFERENCES USED TO WRITE AND JUSTIFY THE SOFTWARE MAINTENANCE DESCRIPTION

The following set of references was chosen to provide coverage of all aspects of software evolution and maintenance. Priority was given to standards, maintenance specific publications, and then general software engineering publications.

References

[AH93] A. Abran and H. Hguyenkim, "Measurement of the Maintenance Process from a Demand-Based Perspective," *Journal of Software Maintenance: Research and Practice*, Vol. 5, no 2, 1993.

[AI98] ANSI/IEEE STD 1061. *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Computer Society Press, 1998.

[Arn92] R.S. Arnold. *Software Reengineering*. IEEE Computer Society, 1992.

[Art88] L.J. Arthur. *Software Evolution: The Software Maintenance Challenge*. John Wiley & Sons, 1988.

[Bas85] V. R. Basili, "Quantitative Evaluation of Software Methodology," *Proceedings First Pan-Pacific Computer Conference,* September 1985.

[Boe81] B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[BBHMMY] C. Boldyreff, E. Burd, R. Hather, R. Mortimer, M. Munro, and E. Younger, "The AMES Approach to Application Understanding: A Case Study," *Proceedings of the International Conference on Software Maintenance-1995*, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[CG90] D.N. Card and R.L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990.

[Dek92] S. M. Dekleva. Delphi Study of Software Maintenance Problems. *Proceedings of the International Conference on Software Maintenance,* 1992.

[DT97] M. Dorfman and R. H. Thayer. *Software Engineering*. IEEE Computer Society Press, 1997.

[GC87] R. B. Grady and D. L. Caswell. *Software Metrics: Establishing a Company-wide Program*. Prentice-Hall, 1987.

[IEEE610.12] IEEE STD 610.2: *IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[IEEE1219] *IEEE STD 1219: Standard for Software Maintenance,* 1998.

[ISO12207] *ISO/IEC 12207: Information Technology-Software Life Cycle Processes, 1995.*

[ISO14764] *ISO/IEC 14764: Software Engineering-Software Maintenance,* 2000.

[Jon98] T.C. Jones. *Estimating Software Costs.* McGraw-Hill, 1998.

[Leh97] M.M. Lehman, Laws of Software Evolution Revisited, EWSPT96, October 1996, LNCS 1149, Springer Verlag, 1997.

[Par86] G. Parikh. *Handbook of Software Maintenance*. John Wiley & Sons, 1986.

[Pfl98] S.L. Pfleeger. *Software Engineering—Theory and Practice.* Prentice Hall, 1998.

[Pig93] T.M. Pigoski, "Maintainable Software: Why You Want It and How to Get It," *Proceedings of the Third Software Engineering Research Forum-November 1993*, University of West Florida Press, Pensacola, FL, 1993.

[Pig97] T.M. Pigoski. *Practical Software Maintenance: Best Practices for Managing your Software Investment.* Wiley, 1997.

[Pre97] R.S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, fourth edition, 1997.

[SKV94] G.E. Stark, L.C. Kern, and C.V. Vowell. *A Software Metric Set for Program Maintenance Management*. Journal of Systems and Software, 1994.

[TG97] A. Takang and P. Grubb. *Software Maintenance Concepts and Practice.* International Thomson Computer Press, 1997.

APPENDIX C – DETAILED BREAKDOWN RATIONALE

Criterion (a): Number of topic breakdowns

One breakdown is provided.

Criterion (b): Reasonableness

The breakdowns are reasonable in that they cover the areas typically discussed in texts and standards, although there is less discussion regarding the pre-maintenance activities, e.g., planning. Other topics such as measures are also often not addressed although they are getting more attention now.

Criterion (c): Generally Accepted

The breakdowns are generally accepted in that they cover the areas typically discussed in texts and standards.

Criterion (d): No specific Application Domains

No specific application domains are assumed.

Criterion (e): Compatibility with Various Schools of Thought

Software maintenance concepts are stable and mature.

Criterion (f): Compatible with Industry, Literature, and Standards

The breakdown was derived from the literature and key standards reflecting consensus opinion. The extent to which industry implements the software maintenance concepts in the literature and in standards varies by company and project.

Criterion (g): As Inclusive as Possible

The primary topics are addressed within the page constraints of the chapter.

Criterion (h): Themes of Quality, Measurement, and Standards

Quality, Measurement and standards are discussed.

Criterion (i): 2 to 3 levels, 5 to 9 topics at the first level

The proposed breakdown satisfies this criterion.

Criterion (j): Topic Names Meaningful Outside the Guide

Wording is meaningful. Version 0.7/0.8 reviews indicated that the wording is meaningful.

Criterion (k) Vincenti Categorization

Topics were applied to the Vincenti Categorization.

Criterion (l): Topics only sufficiently described to allow reader to select appropriate material

A tutorial on maintenance was not provided. Generally accepted concepts were introduced with appropriate references for additional reading were provided.

Criterion (m): Text on the Rationale Underlying the Proposed Breakdowns

The Software Maintenance Theory and Practice was selected as the initial topic in order to introduce the topic. The subtopics are needed to provide definitions and to emphasis why there is a need for maintenance. Categories are critical to understand the underlying meaning of maintenance. All pertinent texts use a similar introduction.

The Maintenance Activities subtopic is needed to differentiate maintenance from development and to show the relationship to other software engineering activities. The subtopic on the Problems of Software Maintenance was chosen to ensure that the software engineers fully comprehended these problems.

Maintenance Process is needed to provide the current references and standards needed to implement the maintenance process.

Every organization is concerned with who will perform maintenance. The Organizational Aspect of Maintenance provides some options. There is always a discussion that maintenance is hard. Every software maintenance reference discusses the fact that maintenance consumes a large portion of the life cycle costs. The topic on Cost and Cost Estimation was provided to ensure that the readers select references to help with this difficult task.

The Software Maintenance Measurements topic is one that is not addressed very well in the literature. Most maintenance books barely touch on the topic. Measurement information is most often found in generalized measurement books. This topic was chosen to highlight the need for unique maintenance measures and to provide specify maintenance measurement references.

The Techniques topic was provided to introduce some of the generally accepted techniques used in maintenance operations.

Finally, there are other resources besides textbooks and periodicals that are useful to software engineers who wish to learn more about software maintenance. This topic is provided to list these additional resources.