SWEBOK Knowledge Area Description for Software Evolution and Maintenance (version 0.5)

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

NOTES FOR REVIEWERS

This section is intended to provide insight to reviewers and will be deleted in the final document.

This is the current draft (version 0.5) of the knowledge area description for software evolution and maintenance. The title of the knowledge area comes from the list of knowledge areas approved by the Industrial Advisory Board for the Guide to the Software Engineering Body of Knowledge Project. This draft builds on version 0.1 and uses a tailored version of the outline provided in the jump-start document. Style and technical guidelines conform to the format for the International Conference on Software Engineering (no more than 10 pages) and follow the IEEE Computer Style Guide. Themes of quality, measurement, tools, and standards are included.

ABSTRACT

This paper presents an overview of the knowledge area of software evolution and maintenance for the Software Engineering Body of Knowledge (SWEBOK) project. A breakdown of topics is presented for the knowledge area along with a short description of each topic. References are given to materials that provide more in-depth coverage of the key areas of software evolution and maintenance. Important knowledge areas of related disciplines are also identified.

Keywords

Software maintenance, software engineering, software evolution, software maintenance measurement, software maintenance planning.

1 INTRODUCTION

This document was prepared in accordance with the "Knowledge Area Description Specifications for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge" dated March 26, 1999. Thus, brief descriptions of the topics are provided so the reader can select the appropriate reference material according to his/her needs.

The topic of this paper is software evolution and maintenance. As many references and common practice is to refer to this topic as maintenance, references in this paper to maintenance refer to software evolution and maintenance.

Software engineering is defined as the application of the systemic, disciplined, quantifiable approach to the development, operation, and maintenance of software. It is the application of engineering to software. The classic life-cycle paradigm for software engineering includes: system engineering, analysis, design, code, testing, and maintenance. This paper addresses the maintenance portion of software engineering and the software life-cycle.

2 BREAKDOWN OF TOPICS FOR SOFTWARE EVOLUTION AND MAINTENANCE

Rationale for the Breakdown

The breakdown of topics for software evolution and 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. They are consistent with what is found in current software engineering literature and standards. The common themes of quality, measurement, tools, and standards are included in the breakdown of topics. The breakdown of topics, along with a rationale for the breakdown, is provided in the next section. The breakdown of topics is contained in Appendix A.

A basic understanding of software evolution and maintenance is needed as the foundation for the remainder of the topics. Thus, the maintenance concepts section provides the definition of maintenance, its basic concepts, and how the concept of system evolution fits into software engineering. Next in importance is what is performed in maintenance. Maintenance activities and roles address the formal types of maintenance and common activities. As with software development, a process is critical to the success and understanding of software evolution and maintenance. Standard maintenance processes are discussed. Appropriate standards are addressed. Organizing for maintenance may be different than for development. Approaches to organizing for maintenance are addressed.

Software evolution and maintenance present unique and different problems for software engineering. These are addressed in the section titled Problems of Software Maintenance. Cost is always a critical topic when discussing software evolution and maintenance. The section on Maintenance Cost and Maintenance Cost Estimation looks at life cycle costs as well as costs for individual evolution and maintenance tasks. Maintenance measurements addresses the topics of quality and metrics. The final topic, tools and techniques for maintenance, aggregates many sub-topics that are not otherwise addressed in this document.

3 BREAKDOWN OF TOPICS

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

Maintenance Concepts

Software maintenance is defined as the modification of a software product after delivery to correct faults, to improve performance, or to adapt the product to a modified environment [1]. A maintainer is an organization that performs maintenance activities [2].

A common perception of maintenance is that it is merely fixing bugs. However, studies over the years have indicated that the majority, over 80%, of the maintenance effort is used for non-corrective actions [3] [4] [5]. This perception is perpetuated by users submitting problem reports that in reality are major enhancements to the system. This “lumping of enhancement requests with problems” contributes to some of the misconceptions regarding maintenance.

The focus of software development is on producing code that implements stated requirements and operates correctly. Maintenance is different than development [6]. 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. Pfleeger [6] states that maintenance has a broader scope, with more to track and control. Thus, configuration management is an important aspect of software evolution and maintenance.

Need for Maintenance

Maintenance is needed to ensure that the system continues to satisfy user requirements. The system changes due to corrective and non-corrective maintenance. According to Martin and McClure [7], maintenance must be performed in order to:

> Correct errors.
>
> Correct design flaws.
>
> Interface with other systems.
>
> Make enhancements.
>
> Make necessary changes to the system.
>
> Make changes in files or databases.
>
> Improve the design.
>
> Convert programs so that different hardware, software, system features, and telecommunications facilities can be used.

The four major aspects that maintenance focuses on are [6]:

> 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.

System Evolution

The previous section indicated that there is a need for maintenance, that software is more than fixing bugs, and that the software evolves. The concept of software evolution and maintenance was first addressed by Lehman in 1980. Simply put he stated that maintenance is really evolutionary developments and that maintenance decisions are aided by understanding what happens to systems over time. His “Five Laws of Software Evolution” clearly depict what happens over time. 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 [6], Sommerville [4], and Arthur [8] have excellent discussions regarding system evolution.

Planning

Software evolution and maintenance should be planned. Whereas developments typically can last for 1-2 years, the operation and maintenance phase, during which software evolution and maintenance occurs, lasts for 5-6 years [6]. Maintenance planning should begin with the decision to develop a new system [5]. 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 is addressed in IEEE 1219 [1] and ISO/IEC [FDIS] 14764 [9]. ISO/IEC [FDIS] 14764 [9] provides guidelines for a maintenance plan.

Maintenance Activities and Roles

Maintenance activities are similar to those of software development. Maintainers perform analysis, design, coding, testing, and documenting. However, for software evolution and maintenance, the activities involve processes unique to maintenance. Maintainers must possess an intimate knowledge of the code's structure and content [6]. Unlike software development, maintainers must perform impact analysis. Analysis is performed in order to determine the cost of making a change. The change request, sometimes called a modification request and often called a problem report, must first be analyzed and translated into software terms [10]. 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.

Activities

Maintainers must perform other activities than those commonly associated with software development. Maintainers must also perform supporting activities such as configuration management (CM), verification and validation, quality assurance, reviews, audits, operating a help desk, and conducting user training. CM is a critical element of the maintenance process [1]. 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 [5] [12] [13]. Maintenance also includes activities such as planning, migration, and retiring of systems [1] [2] [5] [9].

Categories of maintenance

E. B. Swanson of UCLA was one of the first to examine what really happens in maintenance. He believed that by studying the maintenance phase a better understanding of maintenance would result. Swanson was able to create three different categories of maintenance. These are reflected in IEEE 1219 [1], ISO/IEC [FDIS] 14764 [9], and numerous texts**.** Swanson's categories of maintenance and his definitions are as follows:

> Corrective maintenance. Reactive modification of a software product performed after delivery to correct discovered faults.
>
> Adaptive maintenance. Modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment.
>
> Perfective maintenance. Modification of a software product after delivery to improve performance or maintainability.

Common practice is to refer to Adaptive and Perfective maintenance as enhancements. Another type of maintenance, preventive maintenance, is defined in the IEEE Standard on Software Maintenance [1] and the ISO Standard on Software Maintenance [9]. Preventive maintenance is defined as maintenance performed for the purpose of preventing problems before they occur. This type of maintenance could easily fit under corrective maintenance but the international community, and in particular those who are concerned about safety, classify preventive as a separate type of maintenance.

Of note is that Sommerville [4], Pfleeger [6] and others address that the corrective portion of maintenance is only about 20% of the total maintenance effort. The remaining 80% is for enhancements, i.e., the adaptive and perfective categories of maintenance. This further substantiates Lehman's "Five Laws of Software Evolution."

Maintenance Process

The need for software processes is well documented. The Software Engineering Institute's Software Capability Maturity Model (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 CMM applies equally to maintenance and maintainers should have a documented maintenance process

Standards

Schneidewind [11] stressed the need for standardization of maintenance and, as a result, the IEEE Computer Society Software Engineering Standards Subcommittee published the "IEEE Standard for Software Maintenance" [1] in 1993. Later the International Organization for Standards (ISO), developed an international standard for software life-cycle processes, ISO/IEC 12207 [2], which included a maintenance process. ISO/IEC 14764 [9] [FDIS] elaborates the maintenance process of ISO/IEC 12207 [2]. All maintainers should develop maintenance processes bases on these standards. Often these are tailored to meet local requirements.

Maintenance Process Models

Process models provide needed operations and detailed inputs/outputs to those operations. Maintenance process models are provided in ISO 1219 [1] and ISO/IEC 14764 [9].

The maintenance process model described in IEEE 1219 [1], the Standard for Software Maintenance, starts the software maintenance effort during the post-delivery stage and discusses items such as planning for maintenance and metrics outside the process model. That process model with the IEEE maintenance phases is depicted in Figure 2.1.

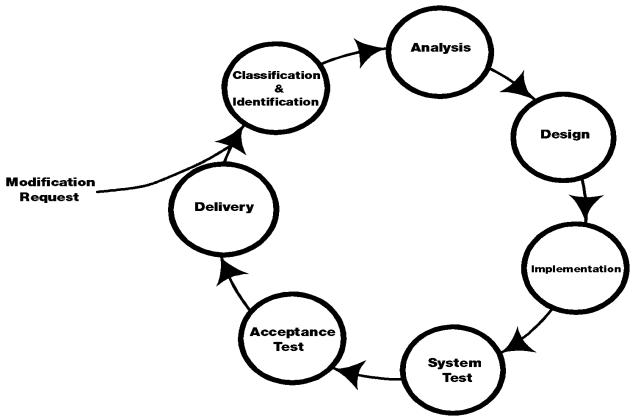

Figure 3.1: The IEEE Maintenance Process

ISO/IEC FDIS 14764 [9] is an elaboration of the maintenance activity from ISO/IEC 12207 [2]. The activities of the maintenance process are similar although they are aggregated a little differently. The maintenance process activities developed by ISO/IEC are shown in Figure 2.2.

Figure 3.2: IEEE Maintenance Process Activities

Takang and Grubb [12] provide a history of maintenance process models leading up to the development of the IEEE and ISO/IEC process models. A good overview of the maintenance process is given by Sommerville [4].

Organization Aspect of Maintenance

The team that develops the software is not always used to maintain the system once it is operational. Often, a separate team 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 [5] [6] [13]. That decision should be made on a case-by-case basis. Outsourcing of maintenance is becoming a major industry. Dorfman and Thayer [10] provide some guidance in the area of outsourcing maintenance. 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 [5], regardless of the organizational structure.

Problems of Software Maintenance

It is important to understand that software evolution and maintenance provide 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. The following discusses some of the technical and management problems relating to software evolution and maintenance.

Technical

Limited Understanding [6]. Several studies 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 extreme interest to maintainers. Pressman [3] relates that it is often difficlt 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.

Testing. The cost of repeating full testing on a major piece of software can be significant in terms of time and money. Thus, determining a sub-sets of tests to perform in order to verify changes are a constant challenge to maintainers [10]. Finding time to test is often difficult [6].

Impact Analysis. The software and the organization must both undergo impact analysis. Critical skills and processes are needed for this area. Impact analysis is necessary for risk abatement.

Management

Alignment with organizational issues. Dorfman and Thayer [10] relate that return on investment is not clear with maintenance. Thus, there is a constant struggle to obtain resources.

Staffing. Maintenance personnel often are viewed as second class citizens [6] and morale suffers [10]. Maintenance is not viewed as glamorous work [3]. Deklava provides a list of staffing related problems based on survey data [5].

Process issues. Maintenance requires several activities that are not found in software development, e.g. help desk support. These present challenges to management [10].

Maintenance Cost and Maintenance Cost Estimation

Maintenance costs are high due to all the problems of maintaining a system [6]. 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 evolution and maintenance.

Cost

Maintenance now consumes a major share of the life cycle costs. Prior to the mid-1980s, the majority of costs went to development. Since that time, maintenance consumes the majority 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. Sommerville [4] and Pfleeger [6] address 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 [8]. It is performed after a change request enters the configuration management process. It is used in concert with the cost estimation techniques discussed below.

Cost estimation

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.

Parametric models

The most significant and authoritative work in the area of parametric models for estimating was performed by Boehm [14]. His COCOMO model, derived from COnstructive COst MOdel, puts the software life cycle and the quantitative life-cycle relationships into a hierarchy of software cost-estimation models [4] [5] [6]. Of significance is that data from past projects is needed in order to use the models. Jones [15] discusses all aspects of estimating costs and provides a detailed chapter on maintenance estimating.

Experience

If a parametric model is not used to estimate maintenance, experience must be used. Sound judgement, reason, a work breakdown structure, educated guesses, and use of empirical/historical data are several approaches. Clearly the best approach is to have empirical data. That data should be provided as a result of a metrics program.

Maintenance Measurements

Software life cycle costs are growing and a strategy for maintenance is needed. Software measurement or software metrics need to be a part of that strategy. Software measurement is the result of a software measurement process [12]. Software metric is often synonymous with software measurement. Grady and Caswell [16] discuss establishing a corporate-wide metrics program. Software metrics are vital for software process improvement but the process must be measurable.

Takang and Grubb [12] state that measurement is undertaken for evaluation, control, assessment, improvement, and prediction. A program must be established with specific goals in mind.

Establishing a metrics program

Successful implementation strategies were used at Hewlett-Packard [16] and at the NASA/Software Engineering Laboratory [5]. Common to many approaches is to use the Goal, Question, Metric (GQM) paradigm put forth by Basili [17]. This approach states that a metric program would consist of: identifying organizational goals; defining the questions relevant to the goals; and then selecting measures that answer the questions.

The *IEEE Standard For a Software Quality Metrics*

Methodology, ANSI/IEEE 1061 992, [18] provides a methodology for establishing quality requirements and identifying, implementing, analyzing and validating process and product software quality metrics. The methodology applies to all software at all phases of any software life cycle and is a valuable resource for software evolution and maintenance.

There are two primary lessons learned from practitioners about metrics programs. The first is to focus on a few key characteristics. The second is not to measure everything. Most organizations collect too much. Thus, a good approach is to evolve a metrics program and to use the GQM paradigm.

Specific Measures

There are metrics that are common to all efforts and the Software Engineering Institute (SEI) identified these as: size; effort; schedule; and quality [5]. Those metrics are a good starting point for a maintainer. ANSI/IEEE 1061 [18] provides examples of metrics together with a complete example of the use of the standard.

Takang and Grubb [12] group metrics into areas of: size; complexity; quality; understandability; maintainability; and cost estimation.

Documentation regarding specific metrics to use in maintenance is not often published. Typically generic software engineering metrics are used and the maintainer determines which ones are appropriate for their organization. IEEE 1219 [1] provides suggested metrics for software programs. Stark, et al [17] provides a suggested list of maintenance metrics used at NASA's Mission Operations Directorate.

Tools and Techniques for Maintenance

This section provides a number of related areas that support software evolution and maintenance. Many other topics, e.g., CM, are orthogonal in nature and are covered in other papers in the Guidebook. Tools are crucial for maintaining large systems. Often tools are integrated into software engineering environments to support the maintenance effort. It is one of the more important supporting areas for maintenance. Program comprehension, also referred to as program understanding, is a necessary technique for performing maintenance. The subtopics of re-engineering, and reverse engineering, are important for legacy systems. These are discussed in the subsections that follow.

Maintenance tools

Takang and Grubb [12] define a software tool as any artifact that supports a maintainer in performing a task. Tools have been in use for years. Debuggers and cross-compilers go back to the early days of computing. Tools have evolved over the years and are well known and used heavily in software development. Tools and related environments are less used in maintenance.

Tools have evolved and are now commonly referred to as Computer-Aided Software Engineering (CASE) tools. Often times these are placed in environments which is defined as all of the automated facilities that a software engineer has available for development or maintenance [4].

Software development and maintenance are specialized activities and need separate systems dedicated to them. Takang and Grubb [12] classify maintenance tools based on the specific tasks they support as follows:

Program understanding and reverse engineering

Testing

Configuration management

Documentation and measurement

For program understanding and reverse engineering, tools include the program slicer, static analyzer, dynamic analyzer, and cross-referencer. Tools that support testing include simulators, test case generators, and test path generators. Tools for configuration must perform code management, version control, and change tracking. Documentation tools include hypertext-based tools, data flow and control chart generators, requirements tracers, and CASE tools [12]. Measurement tools are the same ones used in software development.

IEEE 1219 [1] provides a detailed table of methods and tools mapped to the IEEE maintenance activities.

Program Comprehension

Studies indicate that 40% to 60% of a maintenance programmer's time is spent trying to understand the code. Time is spent in reading and comprehending programs in order to implement changes. Based on the importance of this subtopic, an annual IEEE workshop is now held to address program comprehension [10]. Additional research and experience papers regarding comprehension are found in the annual proceedings of the IEEE Computer Society's International Conference on Software Maintenance (ICSM). Takang and Grubb [12] provide a detailed chapter on comprehension.

Re-engineering

Re-engineering is the examination and alteration of the subject system to reconstitute it in a new form, and the subsequent implementation of the new form. Re-engineering is the most radical (and expensive) form of alteration [10]. It is not undertaken to improve maintainability but is used to replace aging legacy systems. Arnold [19] provides a comprehensive compendium of topics, e.g., concepts, tools and techniques, case studies, and risks and benefits associated with re-engineering.

Reverse engineering

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 redoucumentation. Another type is design recovery [10].

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 [8]. It is performed after a change request enters the configuration management process. Arthur [8] 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.

4 RECOMMENDED REFERENCES FOR SOFTWARE EVOLUTION AND MAINTENANCE

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

[1] *IEEE STD 1219: Standard for Software Maintenance*

[2] *ISO/IEC 12207: Information Technology-Software Life Cycle Processes*

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

[4] I. Sommerville. *Software Engineering.* McGraw-Hill, fifth edition, 1996.

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

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

[7] J. Martin and C. McClure. *Software Maintenance: The Problem and its Solutions.* Prentice-Hall, 1983.

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

[9] *ISO/IEC 14764: [FDIS} Software Engineering-Software Maintenance*

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

[11] N. F. Schneidewind. *The State of Software Maintenance*. IEEE, 1987.

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

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

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

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

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

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

[18] R. S. Arnold. *Software Engineering.* IEEE Computer Society, 1992.

[19] ANSI/IEEE STD 1061. *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Computer Society Press, 1992.

Coverage of the Software Evolution and Maintenance Breakdown topics by the Recommended References.

The cross-reference is shown in Appendix B.

5 KNOWLEDGE AREAS OF RELATED DISCIPLINES

The Guide to the Software Body of Knowledge presents the core Body of Knowledge, i.e., the generally accepted knowledge in the field expected from a graduate with four years of experience. The document "A Baseline for a List of Related Disciplines for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge," dated March 26, 1999 provides a list of potential knowledge areas for software engineers from other disciplines. The following provides a listing of Knowledge

Areas of the Related Disciples that are relevant to the software evolution and maintenance knowledge area. All are keyed to the breakdowns provided in that document.

Computer Science

Foundations

- Complexity analysis
- Complexity classes
- Discrete mathematics
- Program semantics

Algorithms and Data Structures

- Basic data structures
- Abstract data types
- Sorting and searching
- Parallel and distributed algorithms

Computer Architecture

- Digital logic
- Digital systems
- Machine level representation of data
- Number representations
- Assembly level machine organization
- Interfacing and communication
- Alternative architectures
- Digital signal processing
- Performance

Intelligence Systems

- Artificial intelligence
- Agents
- Soft computing

Information Management

- Database models
- Search Engines
- Data mining/warehousing
- Digital libraries
- Transaction processing
- Data compression

Computing at the Interface

- Human-computer interaction
- Graphics
- Vision
- Visualization
- Multimedia
- User-level application generators

Operating Systems

- Tasks, processes and threads
- Process coordination and synchronization
- Scheduling and dispatching
- Physical and virtual memory organizations
- File systems
- Networking fundamentals
- Security
- Protection
- Distributed systems
- Real-time computing
- Embedded systems
- Mobile computing infrastructure

Programming Fundamentals and Skills

- Introduction to programming languages
- Recursive algorithms/programming
- Programming paradigms
- Program-solving strategies
- Compilers/translation
- Code generation

Net-centric Computing

- Computer-supported cooperative work
- Collaboration Technology
- Distributed objects computing
- E-commerce
- Enterprise computing
- Network-level security

Computational Science

- Numerical analysis
- Scientific computing
- Parrallel algorithms
- Modeling and simulation

Social, Ethical, Legal and Professional Issues

- Historical and social context of computing
- Philosophical ethics
- Intellectual property
- Copyrights, patents, and trade secrets
- Risks and liabilities
- Responsibilities of computing professionals
- Computer crime

Mathematics

- Discrete Mathematics
- Calculus
- Probability
- Linear Algebra
- Mathematical Logic

Project Management

- Project Integration Management
- Project Scope Management
- Project Time Management
- Project Quality Management
- Project Human Resource Management
- Project Communications Management
- Project Risk Management
- Project procurement Management

Computer Engineering

- Digital Data Manipulation
- Processor Design
- Digital Systems Design
- Computer Organization
- Storage Devices and Systems
- Peripherals and Communication
- High Performance Systems
- System Design
- Measurement and Instrumentation
- Codes and Standards
- Embedded Systems Software
- Computer Modeling and Simulation

Systems Engineering

Process

- Project Planning
- System breakdown structure
- Design
- Component specification
- Integration
- Maintenance & Operations
- Configuration Management
- Documentation

Essential Functional Processes

- Development
- Test Distribution
- Operations
- Support
- Training
- Disposal

Techniques & Tools

- Metrics
- Privacy
- Process Improvement
- Reliability
- Safety
- Security
- Vocabulary

Management and Management Science

Organizational Environment

- Organizational Characteristics
- Organizational Functions
- Organizational Dynamics

Information Systems Management

- Data Resource Management
- IS Staffing

Training

Management Science

- Optimization
  - Linear Programming
  - Mathematical Programming
- Statistics
- Simulation

Cognitive Science and Human Factors

The Nature of HCI

- (Meta-) Models of HCI

Use and Context of Computers

- Human Social Organization and Work
- Application Areas
- Human-Machine Fit and Adaptation

Human Characteristics

- Language, Communication, Interaction

Computer System and Interface Architecture

- Input and Output Devices
- Computer Graphics

Development Process

Design Approaches

Implementation Techniques

Example Systems and Case Studies

6 SUMMARY

Appendix C identifies the software evolution and maintenance topic areas and the associated Bloom's taxonomy level of understanding on each topic. The levels of understanding from lower to higher are: knowledge, comprehension, application, analysis, synthesis, and evaluation.

Appendix D applies the Vincenti categorization to the software evolution and maintenance topics.

APPENDIX A – SUMMARY OF THE SOFTWARE EVOLUTION AND MAINTENANCE BREAKDOWN

Maintenance Concepts

- *Need for Maintenance*
- *System Evolution*
- *Planning*

Maintenance Activities and Roles

- *Activities*
- *Categories of Maintenance*

Maintenance Process

- *Standards*
- *Maintenance Process Models*

Organization Aspect of Maintenance

Problems of Software Maintenance

- *Technical*
  - Limited Understanding
  - Testing
  - Impact Analysis
- *Management*
  - Alignment with organizational issues
  - Staffing
  - Process issues

Maintenance cost and Maintenance Cost Estimation

- *Cost*
- *Cost estimation*
  - Parametric models
  - Experience

Maintenance Measurements

- *Establishing a Metrics Program*
- *Specific Maintenance Measures*

Tools and Techniques for Maintenance

- *Maintenance Tools*
- *Program Comprehension*
- *Re-engineering*
- *Reverse Engineering*
- *Impact Analysis*

APPENDIX B - COVERAGE OF THE BREAKDOWN TOPICS BY THE RECOMMENDED REFERENCES

TOPIC	REFERENCE																		
	IEEE 1219	**ISO 12207**	**Pre 97**	**Som 96**	**Pig 97**	**Pfl 98**	**MM 83**	**Art 88**	**ISO 14764**	**DT 97**	**Sch 87**	**TG 97**	**Par 86**	**Boe 81**	**Jon 98**	**GC 87**	**SKV 94**	**AI 92**	**Arn 92**
	[1]	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**	**[12]**	**[13]**	**[14]**	**[15]**	**[16]**	**[17]**	**[18]**	**[19]**
Maintenance Concepts	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X		
Need for Maintenance		X		X	X	X	X					X							
System Evolution				X	X	X		X											
Planning	X	X			X	X	X		X										
Maintenance Activities and Roles		X			X	X	X	X		X					X				
Activities	X	X			X	X	X	X	X										
Categories of Maintenance	X	X	X	X	X	X	X	X	X	X		X	X						
Maintenance Process	X	X		X	X	X	X	X	X	X	X	X							
Standards	X	X			X				X	X	X								
Maintenance Process Models	X	X			X				X	X		X							
Organization Aspect of Maintenance			X		X	X	X		X	X		X	X						
Problems of Software Maintenance					X	X						X							
Technical						X				X		X							
Limited Understanding				X		X						X							X
Testing	X							X		X									
Impact Analysis	X					X				X									
Management						X				X		X							
Alignment with organizational issues			X	X		X				X									
Staffing				X	X							X	X						
Process issues										X									

APPENDIX B - COVERAGE OF THE BREAKDOWN TOPICS BY THE RECOMMENDED REFERENCES (Continued)

TOPIC	REFERENCE																		
	IEEE 1219	**ISO 12207**	**Pre 97**	**Som 96**	**Pig 97**	**Pfl 98**	**MM 83**	**Art 88**	**ISO 14764**	**DT 97**	**Sch 87**	**TG 97**	**Par 86**	**Boe 81**	**Jon 98**	**GC 87**	**SKV 87**	**AI 92**	**Arn 92**
	[1]	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**	**[12]**	**[13]**	**[14]**	**[15]**	**[16]**	**[17]**	**[18]**	**[19]**
Maintenance cost and Maintenance Cost Estimation	X				X	X		X	X				X	X					
Cost			X	X	X	X		X						X					
Cost estimation			X	X	X	X								X	X				
Parametric models			X	X	X	X								X	X			X	
Experience	X				X									X					
Maintenance Measurements				X	X	X	X		X			X				X	X	X	
Establishing a Metrics Program					X											X	X		
Specific Maintenance Measures			X	X	X	X	X					X					X	X	
Tools and Techniques for Maintenance												X			X				X
Maintenance Tools	X			X	X	X	X		X			X							X
Program Comprehension					X					X		X							X
Re-engineering	X		X	X				X		X		X			X				X
Reverse Engineering	X		X							X		X			X				X
Impact Analysis						X		X											

APPENDIX C - BLOOM'S TAXONOMY

TOPIC	BLOOM LEVEL
Maintenance Concepts	Comprehension
Need for Maintenance	Comprehension
System Evolution	Comprehension
Planning	Comprehension
Maintenance Activities and Roles	Comprehension
Activities	Comprehension
Categories of Maintenance	Comprehension
Maintenance Process	Synthesis
Standards	Comprehension
Maintenance Process Models	Synthesis
Organization Aspect of Maintenance	Comprehension
Problems of Software Maintenance	Comprehension
Technical	Synthesis
Limited Understanding	Synthesis
Testing	Synthesis
Process issues	Synthesis
Management	Comprehension
Alignment with organizational issues	Comprehension
Staffing	Comprehension
Impact Analysis	Synthesis
Maintenance cost and Maintenance Cost Estimation	Comprehension
Cost	Comprehension
Cost estimation	Synthesis
Parametric models	Synthesis
Experience	Synthesis
Maintenance Measurements	Synthesis
Establishing a Metrics Program	Comprehension
Specific Maintenance Measures	Synthesis
Tools and Techniques for Maintenance	Synthesis
Maintenance Tools	Synthesis
Program Comprehension	Synthesis
Re-engineering	Synthesis
Reverse Engineering	Synthesis
Impact Analysis	Synthesis

APPENDIX D - VINCENTI CATEGORIZATION

FUNDAMENTAL DESIGN CONCEPTS
Maintenance concepts
Need for maintenance
System evolution
Maintenance activities and roles
Activities
Categories of maintenance
Organization aspect of maintenance

CRITERIA AND SPECIFICATIONS
Maintenance process
Standards
Maintenance Measurements
Establishing a metrics program
Specific measures

THEORETICAL TOOLS
Tools and Techniques for Maintenance
Maintenance tools
Program comprehension
Re-engineering
Reverse engineering
Impact Analysis

QUANTITATIVE DATA
Maintenance Cost and Cost Estimation
Cost estimation
Parametric models

PRACTICAL CONSIDERATIONS
Maintenance concepts
Planning
Problems of software maintenance
Technical
Limited understanding
Testing
Impact analysis
Management
Alignment with organizational issues
Staffing
Process issues
Maintenance Cost and Cost Estimation
Cost estimation
Experience

DESIGN INSTRUMENTALITIES
Maintenance Process
Maintenance process models