use of standards of excellence for software engineering practice.

The institute explores promising solutions to potentially significant problems, selects the best candidate solutions, and works on them to determine their value. In some cases, the SEI works to overcome the limitations that prevent a solution from being of general use in the software community. Finally, the SEI moves mature solutions of proven value into widespread use; examples include the Capability Maturity Model® for Software (SW-CMM®), described below, and a model curriculum for a master's degree program in software engineering that has been adopted by universties across the country.

SEI activities are grouped into two principal areas: software engineering management practices and software engineering technical practices.

SOFTWARE ENGINEERING MANAGEMENT PRACTICES

The ability to effectively *manage* the acquisition, development, and evolution of software-intensive systems is a critical requirement of software-intensive systems and thus is emphasized in the SEI program of work. Success in this area increases the ability of software engineering organizations to predict and control quality, schedule, cost, cycle time, and productivity when acquiring, building, and enhancing software systems.

The most widely known aspect of this work is Capability Maturity Modeling®. Capability Maturity Models (CMMs) provide structured, integrated collections of best practices that organizations can use to improve their performance. The SW-CMM is a model for assessing the maturity of the software processes of an organization and for identifying the practices that are required to increase the maturity of these processes. The SW-CMM has become a *de facto* standard for assessing and improving software processes and has been adopted by more than 5,000 organizations worldwide.

The CMM approach for improvement has been applied to disciplines other than software development, such as systems engineering and integrated product and process development (IPPD). As organizations have sought a way to successfully and easily integrate their CMM-based improvement activities, the SEI has collaborated with government and industry organizations to develop a means of integrating maturity models and their associated products (training, assessment instruments, etc.). The Capability Maturity Model Integration (CMMI^{SM}) project, sponsored by the Office of the Under Secretary of Defense and the National Defense Industrial Association (NDIA), is integrating several CMMs into a more general framework to support enterprisewide improvement.

The SEI has also developed the Team Software Process^{SM}, which enables development teams to reduce system-test time and increase software quality by an order of magnitude.

SOFTWARE ENGINEERING TECHNICAL PRACTICES

SEI technical work aims to improve the ability of software engineers to analyze, predict, and control selected functional and nonfunctional properties of software systems. Work is primarily focused on defining, maturing, and accelerating the adoption of improved technical engineering knowledge, processes, and tools.

One SEI initiative concentrates on "survivable systems"—ensuring that appropriate technology and practices are used to prevent successful attacks on networked systems and to limit the damage caused by successful attacks. This work builds on SEI experience with the CERT® Coordination Center (formerly the Computer Emergency Response Team), which counters intrusions into systems connected to the Internet, identifies security flaws that permit intrusions, and works to eliminate those flaws.

Another SEI initiative focuses on techniques for predicting the effect of software architecture decisions on a set of desirable system properties. Still others address a variety of technical issues: identifying and exploiting commonalities that exist across software systems in particular domains, evaluating and integrating commercial off-the-shelf (COTS) components into mission-critical systems while preserving key qualities, and performing incremental and online system upgrades even in the presence of faults caused by the upgrades.

The SEI believes that software organizations will continue to be expected to do more with less. The SEI program will continue to concentrate on key software engineering problems and on facilitating the widespread adoption of improvements that bring management discipling and engineering insight to the practice of software engineering.

STEPHEN E. CROSS
SEI

SOFTWARE ENGINEERING BODY OF KNOWLEDGE PROJECT

Identifying an agreed body of knowledge is an essential step in moving software engineering from an ideal to a recognized profession. Sponsored by the IEEE Computer Society and managed by the Université du Québec à Montréal, the Software Engineering Body of Knowledge (SWEBOK) project is developing a guide to the body of knowledge of software engineering. A trial version of the guide is currently available without charge at http://www.swebok.org. Following an additional two years of trial usage and feedback, the project will revise the Guide.

This article will begin by discussing the desired characteristics of a profession for software engineering. Then the objectives and contents of the Guide are described. Finally, the SWEBOK project and its future will be described.

A SOFTWARE ENGINEERING PROFESSION

In spite of the millions of software professionals worldwide and the ubiquitous presence of software in our society, software engineering has not yet reached the status of a legitimate engineering discipline and a recognized profession. In his Pulitzer-prize-winning book on the history of the medical profession in the Unites States, Starr (1982) states that:

> The legitimization of professional authority involves three distinctive claims: first, that the knowledge and competence of the professional have been validated by a community of his or her peers; second, that this consensually validated knowledge rests on rational, scientific grounds; and third, that the professional's judgment and advice are oriented toward a set of substantive values, such as health. These aspects of legitimacy correspond to the kinds of attributes—collegial, cognitive and moral—usually cited in the term "profession."

Starr's description notes the importance of consensually validated knowledge. How is that knowledge to be applied? Gary Ford and Norman Gibbs (1996) studied several recognized professions including medicine, law, engineering and accounting. They concluded that an engineering profession is characterized by several components:

- An initial *professional education* in a curriculum validated by society through *accreditation*
- Registration of fitness to practice via voluntary *certification* or mandatory *licensing*
- Specialized *skill development* and *continuing professional education*
- Communal support via a *professional society*
- A commitment to norms of conduct often prescribed in a *code of ethics*

The SWEBOK project was formed to address the first three of these components. Articulating a body of knowledge is an essential step toward developing a profession because it represents a broad consensus regarding what a software engineering professional should know. Without such a consensus, no licensing examination can be validated, no curriculum can prepare an individual for an examination, and no criteria can be formulated for accrediting a curriculum. The development of the consensus is also prerequisite to the adoption of coherent skill development and continuing professional education programs in organizations.

ORGANIZATION OF THE SWEBOK GUIDE

The purpose of the SWEBOK Guide is to provide a consensually validated characterization of the bounds of the software engineering discipline and to provide a topical access to the body of knowledge supporting that discipline. The body of knowledge is subdivided into ten knowledge areas and the descriptions of the knowledge areas are designed to discriminate among the various important concepts, permitting readers to find their way quickly to subjects of interest. Upon finding a subject, readers are referred to key papers or book chapters selected because they succinctly present the knowledge.

The content of the SWEBOK Guide is markedly different from computer science. Just as electrical engineering is based upon the science of physics, software engineering should be based on computer science. In both cases, though, the emphasis is necessarily different. Scientists extend our knowledge of the laws of nature while engineers apply those laws of nature to build useful artifacts, under a number of constraints. Therefore, the emphasis of the guide is placed on the construction of useful software artifacts.

Many important aspects of information technology are not covered in the guide, for example, specific programming languages, relational databases and networks. This is a consequence of an engineering-based approach. In all fields—not only computing—the designers of engineering curricula have realized that specific technologies are replaced much more rapidly than the engineering work force. An engineer must be equipped with the essential knowledge that supports the selection of the appropriate technology at the appropriate time in the appropriate circumstance. For example, software systems might be built in FORTRAN using functional decomposition or in C++ using object-oriented techniques. The techniques for integrating and configuring instances of those systems would be quite different. But, the principles and objectives of configuration management remain the same. The SWEBOK Guide therefore does not focus on the rapidly changing technologies, although their general principles are described in relevant knowledge areas.

Therefore, the SWEBOK Guide does not cover all knowledge that is essential to the practice of software engineering. Practicing software engineers will need to know many things about computer science, project management and systems engineering—to name a few—that fall outside the body of knowledge characterized by this guide. However, stating that this information should be known by software engineers is not the same as stating that this knowledge falls within the bounds of the software engineering discipline. The SWEBOK Guide characterizes only the body of knowledge falling within the scope of software engineering.

The emphasis on engineering practice leads the SWEBOK Guide toward a strong relationship with the normative literature. Most of the computer science, information technology and software engineering literature provides information useful to software engineers, but a relatively small portion is normative. A normative document prescribes what an engineer should do in a specified situation rather than providing information that might be helpful. The normative literature is validated by consensus formed among practitioners and is concentrated in standards and related documents. From the beginning, the SWEBOK project was conceived as having a strong relationship to the normative literature of software engineering. The two major standards bodies for software engineering (IEEE Software Engineering Standards Committee and ISO/IEC JTC1/SC7) are represented in the project. [See INTERNATIONAL STANDARDS ORGANIZATIONS, SOFTWARE ENGINEERING STANDARDS, INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE).] Ultimately, it is hoped that software engineering practice standards will contain principles traceable to the SWEBOK Guide. To this end, ISO/IEC JTC1/SC7 has taken steps to initiate the adoption of the Trial Version of the Guide as an ISO/IEC Technical Report.

Objectives of the SWEBOK Project

The SWEBOK Guide should not be confused with the body of knowledge itself. The body of knowledge already exists in the published literature. The purpose of the guide is to describe what portion of the body of knowledge is generally accepted, to organize that portion, and to provide a topical access to it.

The Guide to the Software Engineering Body of Knowledge (SWEBOK) was established with the following five objectives:

1. Promote a consistent view of software engineering worldwide.
2. Clarify the place—and set the boundary—of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics.
3. Characterize the contents of the software engineering discipline.
4. Provide a topical access to the software engineering body of knowledge.
5. Provide a foundation for curriculum development and individual certification and licensing material.

A development process that has engaged approximately 500 reviewers from 42 countries supported the first of these objectives, the consistent worldwide view of software engineering.

The second of the objectives, the desire to set a boundary, motivated the fundamental organization of the SWEBOK Guide. The material that is recognized as being within software engineering is organized into 10 knowledge areas:

- Software requirements
- Software design
- Software construction
- Software testing
- Software configuration management
- Software engineering management
- Software engineering process
- Software engineering tools and methods
- Software maintenance
- Software quality

Each of the 10 knowledge areas is described in a chapter of the SWEBOK Guide.

In establishing a boundary for software engineering, it is also important to identify the other disciplines that share a boundary and often a common intersection with software engineering. To this end, the guide also recognizes seven related disciplines:

- Cognitive sciences and human factors
- Computer engineering
- Computer science
- Management and management science
- Mathematics
- Project management
- Systems engineering

Of course, software engineers should know material from these fields. However, it is not an objective of the SWEBOK Guide to characterize the knowledge of the related disciplines.

Hierarchical Organization of the Guide

The organization of the knowledge area descriptions or chapters, shown in Figure 1, supports the third of the project's objectives—a characterization of the contents of software engineering.

The SWEBOK Guide uses a hierarchical organization to decompose each knowledge area into a set of topics with recognizable labels. A two- or three-level breakdown provides a reasonable way to find topics of interest. The guide treats the selected topics in a manner compatible with major schools of thought and with breakdowns generally found in industry and in software engineering literature and standards. The breakdowns of topics do not presume particular application domains, business uses, management philosophies, development methods, and so forth. The extent of each topic's description is only that needed to understand the generally accepted nature of the topics and for the reader to successfully find reference material. After all, the body of knowledge is found in the reference materials, not in the guide itself.

References to the Literature

To provide a topical access to the knowledge—the fourth of the project's objectives—the SWEBOK Guide identifies reference materials for each knowledge area including book chapters, refereed papers, or other well-recognized sources of authoritative information. Each knowledge area description also includes a matrix that relates the

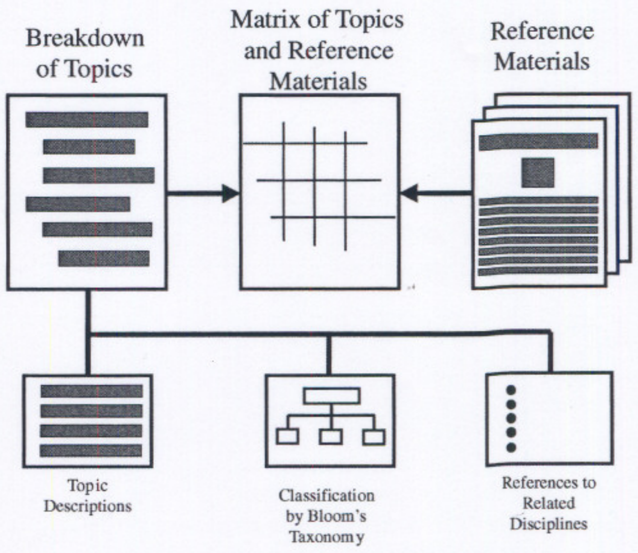
Breakdown of Topics
Matrix of Topics and Reference Materials
Reference Materials
Topic Descriptions
Classification by Bloom's Taxonomy
References to Related Disciplines

Figure 1. Organization of knowledge areas.

reference materials to the listed topics. The total volume of cited literature is intended to be suitable for mastery through the completion of an undergraduate education plus four years of experience.

It should be noted that the guide is not comprehensive in its citations. Much material that is both suitable and excellent is not referenced. Materials were selected because they are written in English, readily available, easily readable, and—taken as a whole—provide coverage of the described topics.

Depth of Treatment

In its depth of treatment, the SWEBOK guide follows an approach that supports the fifth of the project's objectives—providing a foundation for curriculum development, certification, and licensing. It applies a criterion of *generally accepted* knowledge, which is distinguished from advanced and research knowledge (on the grounds of maturity) and from specialized knowledge (on the grounds of generality of application). A second definition of *generally accepted* comes from the PMI: "The generally accepted knowledge applies to most projects most of the time, and widespread consensus validates its value and effectiveness (Project Management Institute, 1996).

However, generally accepted knowledge does not imply that one should apply the designated knowledge uniformly to all software engineering endeavors—each project's needs determine that—but it does imply that competent, capable software engineers should be equipped with this knowledge for potential application. Additionally, the knowledge area descriptions are somewhat forward-looking—the guide considers not only what is generally-accepted today but also what could be generally accepted in three to five years.

Ratings

As an aid notably, to curriculum developers, and in support of the project's fifth objective, the SWEBOK guide rates each topic with one of a set of pedagogical categories commonly attributed to Benjamin Bloom (1956). The concept is that educational objectives can be classified into six categories representing increasing depth: knowledge, comprehension, application, analysis, synthesis, and evaluation.

References to Related Disciplines

The knowledge area descriptions may also contain references to subjects within the seven related disciplines.

OVERVIEW OF THE SWEBOK GUIDE

Figure 2 maps the 10 knowledge areas and the important topics incorporated within them. The first five knowledge areas are presented in traditional waterfall life-cycle sequence. The subsequent knowledge areas are presented in alphabetical order. This is identical to the sequence in which they are presented in the SWEBOK Guide. Brief summaries of the knowledge area descriptions appear in the next section.

Software Requirements

A *requirement* is defined as a property that must be exhibited in order to solve some problem of the real world.

The first knowledge subarea introduces the *requirements engineering process*, orienting the remaining five topics and showing how requirements engineering dovetails with the overall software engineering process. It describes process models, process actors, process support and management and process quality improvement.

The second subarea is *requirements elicitation*, which is concerned with where requirements come from and how the requirements engineer can collect them. It includes requirement sources and techniques for elicitation.

The third subarea, *requirements analysis*, is concerned with the process of analyzing requirements to:

- Detect and resolve conflicts between requirements
- Discover the bounds of the system and how it must interact with its environment
- Elaborate system requirements to software requirements.

Requirements analysis includes requirements classification, conceptual modeling, architectural design, requirements allocation, and requirements negotiation.

The fourth subarea is *software requirements specification*. It describes the structure, quality and verifiability of the requirements document. This may take the form of two documents, or two parts of the same document with different readership and purposes. The first document is the system requirements definition document, and the second is the software requirements specification. The subarea also describes the document structure and standards and document quality.

The fifth sub-area is *requirements validation*, whose aim is to pick up any problems before resources are committed to addressing the requirements. Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right system (i.e., the system that the user expects). It is subdivided into descriptions of the conduct of requirements reviews, prototyping, model validation, and acceptance tests.

The last sub-area is *requirements management*, which is an activity that spans the whole software life cycle. It is fundamentally about change management and the maintenance of the requirements in a state that accurately mirrors the software to be, or that has been, built. It includes change management, requirements attributes and requirements tracing (see REQUIREMENTS MANAGEMENT).

Software Design

Software design is an activity that spans the whole software life-cycle (see DESIGN). The knowledge area is divided into six subareas.

The first one presents the *basic concepts* and notions that form an underlying basis to the understanding of the role and scope of software design. These are general

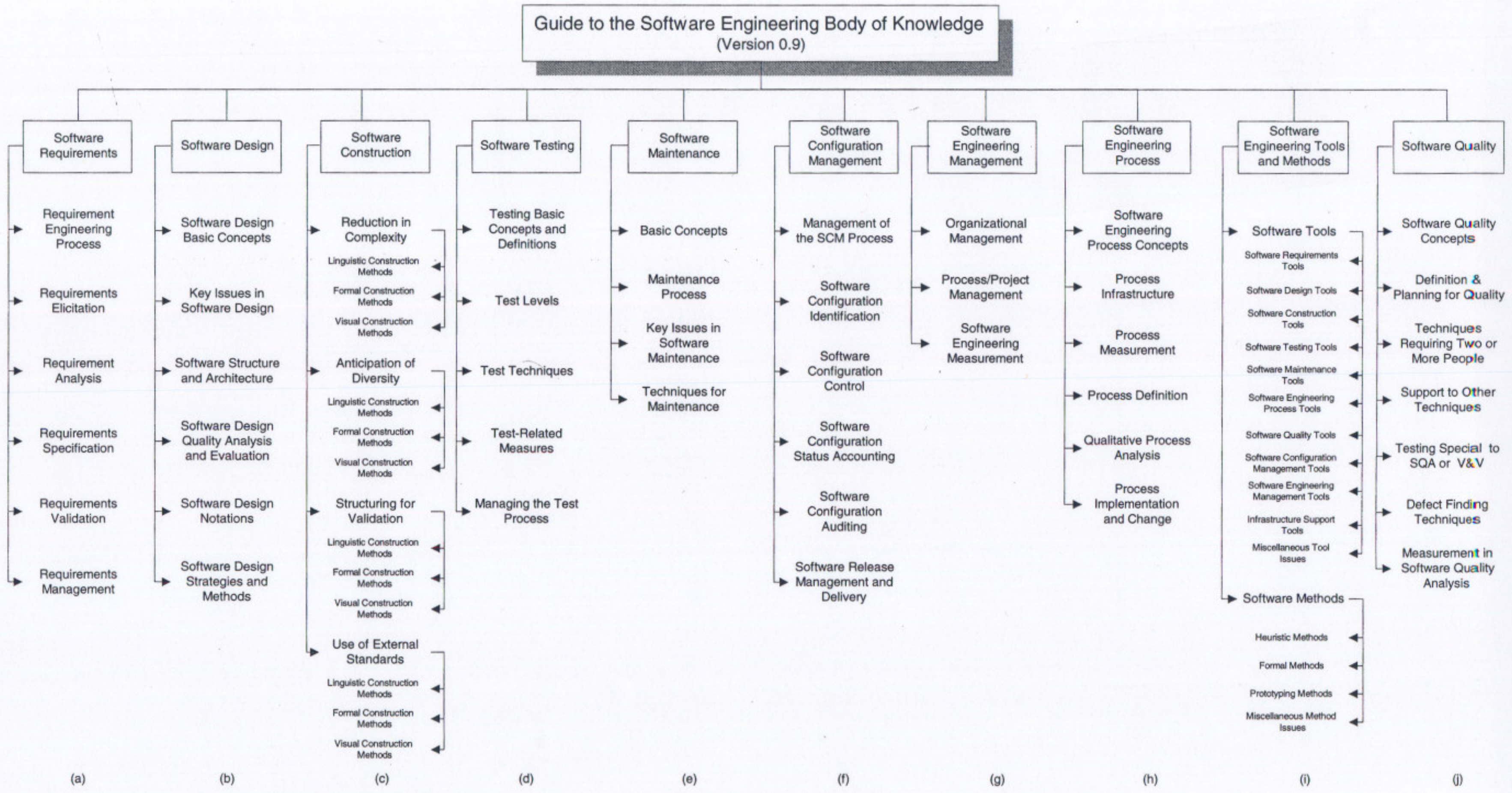
Guide to the Software Engineering Body of Knowledge
(Version 0.9)
Software Requirements
Requirement Engineering Process
Requirements Elicitation
Requirement Analysis
Requirements Specification
Requirements Validation
Requirements Management
Software Design
Software Design Basic Concepts
Key Issues in Software Design
Software Structure and Architecture
Software Design Quality Analysis and Evaluation
Software Design Notations
Software Design Strategies and Methods
Software Construction
Reduction in Complexity
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Anticipation of Diversity
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Structuring for Validation
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Use of External Standards
Linguistic Construction Methods
Formal Construction Methods
Visual Construction Methods
Software Testing
Testing Basic Concepts and Definitions
Test Levels
Test Techniques
Test-Related Measures
Managing the Test Process
Software Maintenance
Basic Concepts
Maintenance Process
Key Issues in Software Maintenance
Techniques for Maintenance
Software Configuration Management
Management of the SCM Process
Software Configuration Identification
Software Configuration Control
Software Configuration Status Accounting
Software Configuration Auditing
Software Release Management and Delivery
Software Engineering Management
Organizational Management
Process/Project Management
Software Engineering Measurement
Software Engineering Process
Software Engineering Process Concepts
Process Infrastructure
Process Measurement
Process Definition
Qualitative Process Analysis
Process Implementation and Change
Software Engineering Tools and Methods
Software Tools
Software Requirements Tools
Software Design Tools
Software Construction Tools
Software Testing Tools
Software Maintenance Tools
Software Engineering Process Tools
Software Quality Tools
Software Configuration Management Tools
Software Engineering Management Tools
Infrastructure Support Tools
Miscellaneous Tool Issues
Software Methods
Heuristic Methods
Formal Methods
Prototyping Methods
Miscellaneous Method Issues
Software Quality
Software Quality Concepts
Definition & Planning for Quality
Techniques Requiring Two or More People
Support to Other Techniques
Testing Special to SQA or V&V
Defect Finding Techniques
Measurement in Software Quality Analysis
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)

Figure 2. Breakdown of knowledge areas in the SWEBOK guide. (© IEEE—Stoneman (version 0.9)—February 2001.)

concepts, the context of software design, the design process and the enabling techniques for software design.

The second subarea presents the *key issues of software design*. They include concurrency, control and handling of events, distribution, error and exception handling, interactive systems and persistence.

The third subarea is *software structure and architecture*, in particular architectural structures and viewpoints, architectural styles, design patterns, and finally families of programs and frameworks.

The fourth subarea describes *software design quality analysis and evaluation*. While a whole knowledge area is devoted to software quality, this subarea presents the topics more specifically related to software design. These aspects are quality attributes, quality analysis, and evaluation tools and measures.

The fifth one is *software design notations*, which are divided into structural and behavioral descriptions.

The last subarea covers *software design strategies and methods*. First, general strategies are described, followed by function-oriented methods, then object-oriented methods, data-structure centered design and a group of other methods, such as formal and transformational methods.

Software Construction

Software construction is a fundamental act of software engineering: the construction of working meaningful software through a combination of coding, validation, and (unit) testing.

The first and most important method of breaking the subject of software construction into smaller units is to recognize the four principles that most strongly affect the way in which software is constructed. These principles are the *reduction of complexity*, the *anticipation of diversity*, the *structuring for validation* and the *use of external standards*.

A second and less important method of breaking the subject of software construction into smaller units is to recognize three styles/methods of software construction, namely, *linguistic*, *formal* and *visual*.

A synthesis of these two views is presented and techniques are classified by principle and approach.

Software Testing

Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the specified expected behavior (see TEST MANAGEMENT AND ORGANIZATION). It includes five subareas.

It begins with a description of *basic concepts*. First, the testing terminology is presented, then the theoretical foundations of testing are described, with the relationship of testing to other activities.

The second subarea is the *test levels* dealing with testing corresponding to levels of integration as well as testing for specific objectives.

The third subarea describes the *test techniques* themselves. A first category describes tests based on some specified base of material; a second category describes tests that are implemented in ignorance of the implementation. A discussion of how to select and combine the appropriate techniques is presented.

The fourth subarea covers *test-related measures*. The measures are grouped into those related to the evaluation of the program under test and the evaluation of the tests performed.

The last subarea describes the *management* specific to the test process. It includes management concerns and the test activities.

Software Maintenance

Once in operation, anomalies are uncovered, operating environments change, and new user requirements surface (see MAINTENANCE). The maintenance phase of the life cycle commences upon delivery but maintenance activities occur much earlier. The software maintenance knowledge area is divided into four subareas.

The first one presents the domain's *basic concepts*, including definitions, the main activities, and the problems of software maintenance.

The second sub-area describes the *maintenance process*, based on the standards IEEE 1219 and ISO/IEC 14764.

The third sub-area treats *key issues* related to software maintenance. The topics covered are technical, management, cost and estimation, and measurement issues.

Techniques for maintenance constitute the fourth subarea. Those techniques include program comprehension, reengineering, reverse engineering, and impact analysis.

Software Configuration Management

Software configuration management (SCM) is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle (see CONFIGURATION MANAGEMENT). This knowledge area includes six subareas.

The first subarea is the *management of the SCM process*. It covers the topics of the organizational context for SCM, constraints and guidance for SCM, planning for SCM, the SCM plan itself, and surveillance of SCM.

The second sub-area is *software configuration identification*, which identifies items to be controlled, establishes identification schemes for the items and their versions, and establishes the tools and techniques to be used in acquiring and managing controlled items. The topics in this subarea are the identification of the items to be controlled and the software library.

The third sub-area is the *software configuration control*, which is the management of changes during the software life cycle. The topics are, (**1**) requesting, evaluating, and approving software changes; (**2**) implementing software changes; and (**3**) deviations and waivers.

The fourth subarea is *software configuration status accounting*. Its topics are software configuration status information and status reporting.

The fifth sub-area is *software configuration auditing*, consisting of software functional configuration auditing, software physical configuration auditing, and in-process audits of a software baseline.

The last subarea is *software release management and delivery*, covering software building and software release management.

Software Engineering Management

While it is true to say that in one sense it should be possible to manage software engineering in the same way as any other (complex) process, there are aspects particular to software products and the software engineering process that complicate effective management (see PROJECT MANAGEMENT). There are three subareas for software engineering management.

The first is *organizational management*, comprising policy management, personnel management, communication management, portfolio management, and procurement management.

The second subarea is *process/project management*, including initiation and scope definition, planning, enactment, review and evaluation, and closure.

The third and last subarea is *software engineering measurement*, where general principles about software measurement are covered. The first topics presented are the goals of a measurement program, followed by measurement selection, measuring software and its development, collection of data and, finally, software measurement models.

Software Engineering Process

The software engineering process knowledge area is concerned with the definition, implementation, measurement, management, change and improvement of the software engineering process itself (see CAPABILITY MATURITY MODEL FOR SOFTWARE). It is divided into six sub-areas.

The first one presents the *basic concepts*: themes and terminology.

The second subarea is *process infrastructure*, where the software engineering process group concept is described, as well as the experience factory.

The third subarea deals with *measurements specific to software engineering process*. It presents the methodology and measurement paradigms in the field.

The fourth subarea describes knowledge related to *process definition*: the various types of process definitions, the life-cycle framework models, the software life-cycle models, the notations used to represent these definitions, process definitions methods, and automation relative to the various definitions.

The fifth subarea presents *qualitative process analysis*, especially the process definition review and root cause analysis.

Finally, the sixth subarea concludes with *process implementation and change*. It describes the paradigms and guidelines for process implementation and change, and the evaluation of the outcome of implementation and change.

Software Engineering Tools and Methods

The software engineering tools and methods knowledge area includes both the software development environments and the development methods knowledge areas identified in the Strawman version of the guide (see SOFTWARE TOOLS]

Software development environments are the computer-based tools that are intended to assist the software development process. Development methods impose structure on the software development activity with the goal of making the activity systematic and ultimately more likely to be successful.

The partitioning of the *software tools* section uses the same structure as the Trial version of the Guide to the Software Engineering Body of Knowledge. The first nine subsections correspond to *the other nine knowledge areas*. Two additional subsections are provided: one for *infrastructure support tools* that do not fit in any of the earlier sections, and a *miscellaneous* subsection for topics, such as tool integration techniques, that are potentially applicable to all classes of tools.

The *software development methods* section is divided into four subsections: heuristic methods dealing with informal approaches, formal methods dealing with mathematically based approaches, prototyping methods dealing with software development approaches based on various forms of prototyping, and miscellaneous method issues.

Software Quality

The final chapter deals with software quality considerations that transcend the life-cycle processes. Software quality is a ubiquitous concern in software engineering, so it is considered in many of the other knowledge areas and the reader will notice pointers to those knowledge areas through this knowledge area (see QUALITY ASSURANCE). The knowledge area description covers four subareas.

The first subarea describes the *software quality concepts* such as measuring the value of quality, the ISO/IEC 9126 quality model, dependability, and other special types of system and quality needs.

The second subarea covers the *purpose and planning* of software quality assurance (SQA) and V&V (verification and validation). It includes common planning activities, and both the SQA and V&V plans.

The third subarea describes the *activities and techniques* for SQA and V&V. It includes static and dynamic techniques as well as other SQA and V&V testing.

The fourth subarea describes *measurement* applied to SQA and V&V. It includes the fundamentals of measurement, measures, measurement analysis techniques, defect characterization, and additional uses of SQA and V&V data.

THE SWEBOK PROJECT

From 1993 to 2000, the IEEE Computer Society and the Association for Computing Machinery (ACM) cooperated in promoting the professionalization of software engineering through their joint Software Engineering Coordinating Committee (SWECC). One important product of the SWECC was a Code of Ethics completed through volunteer efforts in 1997. The SWEBOK project was initiated by the SWECC in 1998, although the ACM subsequently decided to withdraw from participation. The SWEBOK project's scope, the variety of communities involved, and the need for broad participation suggested a need for full-time rather than volunteer management. For this purpose, the IEEE Computer Society contracted the Software Engineering Management Research Laboratory at the Université du Québec à Montréal (UQAM) to manage the effort.

The project team consists of personnel from the IEEE Computer Society and from UQAM. Leonard Tripp (Boeing), the 1999 President of the Computer Society, is the Chair of the SWECC. Alain Abran (UQAM) and James W. Moore (The MITRE Corporation) serve as Executive Editors, with primary responsibilities for representing the project. Pierre Bourque (École de Technologie Supérieure) and Robert Dupuis (UQAM) are the project editors, with primary responsibility for managing the project.

Like any software project, the SWEBOK project has many stakeholders—some of whom are formally represented. An Industrial Advisory Board—composed of the IEEE Computer Society, representatives from industry (Boeing, the MITRE Corporation, Rational Software, Raytheon Systems, and SAP Labs-Canada), and research agencies (National Institute of Standards and Technology, National Research Council of Canada)—has provided financial support for the project. The financial support permits making the products of the SWEBOK project publicly available without any charge. IAB membership is supplemented with the chair of the appropriate international standards committee, ISO/IEC JTC1/SC7, and the chair of the Computing Curricula 2001 initiative. The IAB reviews and approves the project plans, oversees consensus building and review processes, promotes the project, and lends credibility to the effort. In general, it ensures the relevance of the effort to real-world needs. From the outset, it was understood that an implicit body of knowledge already exists in textbooks on software engineering. To ensure taking advantage of existing literature, Steve McConnell, Roger Pressman, and Ian Sommerville—the authors of three best-selling textbooks on software engineering—served on a Panel of Experts to provide advice on the initial formulation of the project and the structure of the SWEBOK Guide. In all cases, the project sought international participation to maintain a broad scope of relevance.

The project team developed two important principles for guiding the project: *transparency* and *consensus*. Transparency implies that the development process is itself documented, published, and publicized so that important decisions and status are visible to all concerned parties. Consensus implies that the practical method for legitimizing a statement of this kind is through broad participation and agreement by all significant sectors of the relevant community.

The project plan includes three successive phases: Strawman, Stoneman, and Ironman. An early prototype, Strawman, demonstrated how the project might be organized. The publication of the current trial version of the SWEBOK Guide marks the end of the Stoneman phase of the project. Development of the Ironman version will commence after trial application of the current SWEBOK Guide.

The project team organized the development of the Stoneman phase into three public review cycles. The first review cycle focused on the soundness of the proposed breakdown of topics within each knowledge area. The second review looked at the contents of the guide from important viewpoints (e.g., educator, practitioner). The third review focused on the coherency of the guide as a whole. In all, roughly 500 reviewers–about half from the United States and the remainder from 41 other countries—have provided nearly 10,000 comments. All review material and comments are available on the project Web site.

FUTURE PLANS

The current trial version of the SWEBOK Guide represents an enormous step in consensus building, but is not yet proved by trial usage. The results of a two-year period of field testing will be considered by the project team in formulating plans for the Ironman phase of the project. During the Ironman phase, the document will be revised and reviewed through additional rounds of consensus formation.

Already some changes seem likely. An additional knowledge area for component integration might directly address software reuse and COTS component integration. Some reviewers have suggested a knowledge area for human–computer interface. The principles of the software quality knowledge area might be distributed among the other knowledge areas in order to achieve a tighter relationship. The description in the current tools and methods knowledge area is already organized by knowledge area; a logical next step might be to distribute the materials among the referenced knowledge areas. Finally, some nascent knowledge areas may emerge. In three years, a community consensus on the nature of software architecture might lead to the development of a distinct knowledge area.

ACKNOWLEDGMENTS

Portions of this article are adapted, with permission, from Guide to the Software Engineering Body of Knowledge, Trial version (version 0.9), February 2001, © Copyright 2001, Institute of Electrical and Electronics Engineers.

BIBLIOGRAPHY

B. S. Bloom, ed., *Taxonomy of Educational Objectives; The Classification of Educational Goals*, Handbook 1, Longmans, Green, New York and Toronto, 1956.

G. Ford and N. E. Gibbs, *A Mature Profession of Software Engineering*, Technical report CMU/SEI-96-TR-004, Software Engineering Institute/Carnegie-Mellon University, Pittsburgh, PA, 1996.

Project Management Institute, *A Guide to the Project Management Body of Knowledge*, Project Management Institute, Upper Darby, PA, 1996. Available at: *http://www.pmi.org/publictn/pmboktoc.htm/*.

P. Starr, *The Social Tranformation of Amercian Medicine*, Basic Books, New York, 1982, p. 15.

JAMES W. MOORE
The MITRE Corporation
PIERRE BOURQUE
École de Technologie Supérieure
ROBERT DUPUIS
ALAIN ABRAN
Université du Québec à Montreal
LEONARD TRIPP
Boeing

SOFTWARE ENGINEERING ENVIRONMENTS

INTRODUCTION

A software factory is meant to be capable of providing computer support for the coordinated work of software developers in large software development projects. The term *software factory* hence denotes a number of things: people and their respective roles in software development; computer supported tools and their combined use in software development; and a co-ordination process model to guide people in their proper use of tools and in their proper joined work integration as depicted below (see Fig. 1).

Recent developments have led from closed environments, comprising fixed sets of tightly coupled tools for specific phases, to open environments that enable the plugging of new tools as the requirements evolve (see also SOFTWARE FACTORY).

To cope with this new dimension in CASE technology, standards organizations intend to support the effort with reference models; for example, the Information Systems Engineering Reference Model (ISE/RM) of the ISO; the Standards Manual of the Object Management Group (OMG), ECMA's Reference Model for Computer Assisted Software Engineering Environment Frameworks; ECMA's Support Environment for Open Distributed Processing (SE/OPD); and OSF's Distributed Computing Environment (OSF/DCE).

The concepts presented in this article have been primarily developed in the Eureka Software Factory (ESF) project. This article hence carries the flavor of that project in the way it introduces software factory concepts and in the way it explains these concepts in the larger context of computer-aided software development (see also EUREKA SOFTWARE FACTORY).

Computer support in a software factory will be provided by a software system that is called *factory support environment* (FSE). In order to support software development in the manner outlined above, the FSE is needed to provide a number of integration services. FSE integration services will be explained first by introducing a conceptual view of a software factory and later on as an architectural view. The conceptual view, called the software factory core, introduces a number of integration stages: interworking, interaction, interoperation, and interconnection. The architectural view introduces the mechanisms that support integration at the respective stages.

THE SOFTWARE FACTORY CORE

What the ESF CoRe Contains

The ESF CoRe is a road map of the Software Factory domain. As such, it embodies a view of what industrial software production means: what sorts of entities and events ought to be distinguished in software factories, and how these relate to each other. Concretely, the CoRe consists of a set of *definitions,* some *arguments* as to what those definitions imply, and some *pictures* showing how the definitions are interrelated.

The structure of the CoRe derives from a strategic decision to view industrial software production as supported by layered communication processes. Two views result:

- A *conceptual* view, using an extended OSI layer model to distinguish levels of factory communication and the kinds of work that can be conducted at each level.
- An *architectural* view, which categorizes the entities that communicate through and across each of the layers, and places them in an evolutionary context.

The conceptual view hence introduces an extended reference model for factory communication and the architectural view the essential ingredients of a software factory support environment.

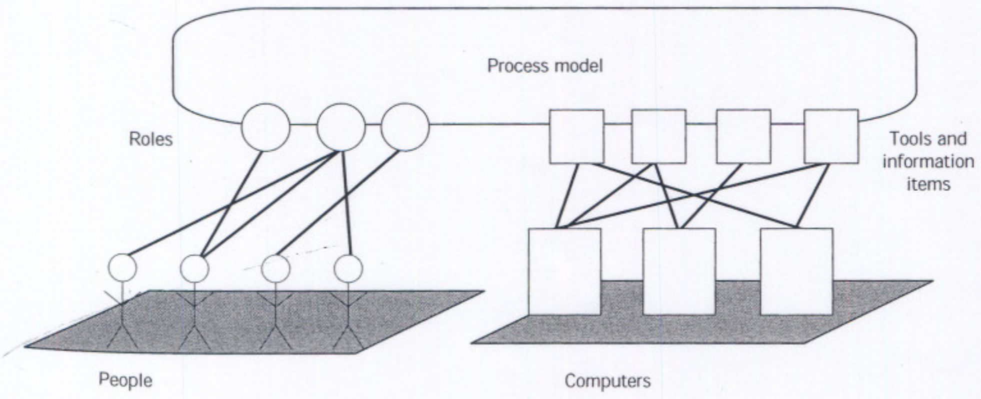

Figure 1. Software factory model.

ENCYCLOPEDIA OF SOFTWARE ENGINEERING

SECOND EDITION

VOLUME 2

John J. Marciniak, ***Editor-in-chief***

A Wiley-Interscience Publication
John Wiley & Sons, Inc.

This book is printed on acid-free paper. ♾

Library of Congress Cataloging-in-Publication Data is available.

Marciniak, John J.
  Encyclopedia of Software Engineering, Second Edition

ISBN - Volume 1	0-471-21008-0
ISBN - Volume 2	0-471-21007-2
ISBN - Set	0-471-37737-6

Printed in the United States of America.

10 9 8 7 6 5 4 3 2