Reporting on the SWEBOK project, the authors—who represent the project's editorial team—discuss the three-phase plan to characterize a body of knowledge, a vital step toward developing software engineering as a profession.

The Guide to the Software Engineering Body of Knowledge

Pierre Bourque, Robert Dupuis, and Alain Abran,
University of Quebec at Montreal
James W. Moore, The MITRE Corporation
Leonard Tripp, The Boeing Company

The IEEE Computer Society and the Association for Computing Machinery are working on a joint project to develop a guide to the Software Engineering Body of Knowledge (SWEBOK). Articulating a body of knowledge is an essential step toward developing a profession because it represents a broad consensus regarding the contents of the discipline. Without such a consensus, there is no way to validate a licensing examination, set a curriculum to prepare individuals for the examination, or formulate criteria for accrediting the curriculum.

The SWEBOK project (http://www.swebok.org) is now nearing the end of the second of its three phases. Here we summarize the results to date and provide an overview of the project and its status.

Table 1. The SWEBOK knowledge areas and their corresponding specialists.

Knowledge Area	Specialists
Software configuration management	John A. Scott and David Nisse, Lawrence Livermore Laboratory, US
Software construction	Terry Bollinger, The Mitre Corporation, US
Software design	Guy Tremblay, Université du Québec à Montréal, Canada
Software engineering infrastructure	David Carrington, The University of Queensland, Australia
Software engineering management	Stephen G. MacDonell and Andrew R. Gray, University of Otago, New Zealand
Software engineering process	Khaled El Emam, National Research Council, Canada
Software evolution and maintenance	Thomas M. Pigoski, Techsoft, US
Software quality analysis	Dolores Wallace and Larry Reeker, National Institute of Standards and Technology, US
Software requirements analysis	Pete Sawyer and Gerald Kotonya, Lancaster University, UK
Software testing	Antonia Bertolino, National Research Council, Italy

Table 2. Related Disciplines.

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

Objectives and Audience

The SWEBOK project team established the project with five objectives:

1. Characterize the contents of the software engineering discipline.
2. Provide topical access to the software engineering body of knowledge.
3. Promote a consistent view of software engineering worldwide.
4. 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.
5. Provide a foundation for curriculum development and individual certification material.

The product of the SWEBOK project will not be the body of knowledge itself, but rather a guide to it. The knowledge already exists; our goal is to gain consensus on the core subset of knowledge characterizing the software engineering discipline.

To achieve these goals, we oriented the project toward a variety of audiences. It aims to serve public and private organizations in need of a consistent view of software engineering for defining education and training requirements, classifying jobs, and developing performance evaluation policies. It also addresses practicing software engineers and the officials responsible for making public policy regarding licensing and professional guidelines. In addition, professional societies and educators defining the certification rules, accreditation policies for university curricula, and guidelines for professional practice, as well as students learning the software engineering profession, will benefit from SWEBOK.

The Guide

The project comprises three phases: Strawman, Stoneman, and Ironman. The Strawman guide, completed within nine months of project initiation, served as a model for organizing the SWEBOK guide.[1] Spring 2000 will see the completion of the Stoneman version, after which we'll commence the Ironman phase, which will continue for two or three years. Following the principles of the Stoneman phase, Ironman will benefit from more in-depth analyses, a broader review process, and the experience gained from trial usage.

The SWEBOK Guide will organize the body of knowledge into several Knowledge Areas. In its current draft, the Stoneman version of the Guide identifies 10 KAs—Table 1 lists the KA specialists responsible for preparing each DA description. In addition, we're considering seven related disciplines (see Table 2).

The distinction between KAs and related disciplines is important to the Guide's purpose. The project will specify KAs—and topics within these KAs—that are regarded as core knowledge for software engineers. Software engineers should also know material from the related disciplines, but the SWEBOK project will not attempt to specify that material. Instead, we're leaving that to other efforts such as those being coordinated by the Joint IEEE Computer Society and ACM Software Engineering Coordinating Committee, or the Working Group on Software Engineering Education.[2]

As Figure 1 shows (and as the following sections explain), each KA description—which should be around 10 pages—contains several important components.

Hierarchical organization

The SWEBOK Guide will use a hierarchical organization to decompose each KA into a set of topics with recognizable labels. A two- or three-level breakdown will provide a reasonable way for readers to find topics of interest. The Guide will treat 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 breakdown of topics will not presume particular application domains, business uses, management philosophies, development methods, and so forth. The extent of each topic's description will be only that needed 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.

From the outset, the question arose as to the depth of treatment the Guide should provide. After substantial discussion, we adopted a concept of *generally accepted* knowledge,[3] which we had to distinguish from advanced and research knowledge (on the grounds of maturity) and from specialized knowledge (on the grounds of generality of application). The generally accepted knowledge applies to most projects most of the time, and widespread consensus validates its value and effectiveness.

However, generally accepted knowledge does not imply that we 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. More precisely, generally accepted knowledge should be included in the study material for a software engineering licensing examination that graduates would take after gaining four years of work experience. Although this criterion is specific to the US style of education and does not necessarily apply to other countries, we deem it useful. However, both definitions of generally accepted knowledge should be seen as complementary.

Additionally, the proposed breakdown must be somewhat forward-looking—we're considering not only what is generally accepted today but also what will be generally accepted in three to five years.

Reference materials and a matrix

The Guide will identify reference materials for each KA. They might be book chapters, refereed papers, or any other well-recognized source of authoritative information—but the reference should be written in English and generally available. We prefer material to which the IEEE Computer Society or the ACM already has publication rights because we want to make the references available on the Internet without charge.

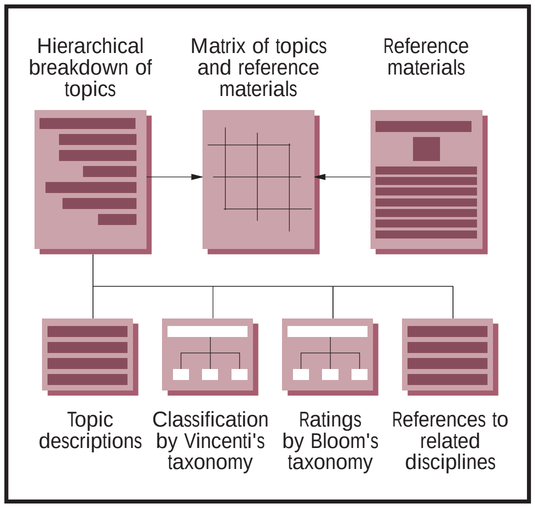

Hierarchical breakdown of topics
Matrix of topics and reference materials
Reference materials
Topic descriptions
Classification by Vincenti's taxonomy
Ratings by Bloom's taxonomy
References to related disciplines

Figure 1. The organization of a Knowledge Area description.

The Guide will also include a matrix that relates the reference materials to the listed topics. Of course, a particular reference might apply to more than one topic.

Classification

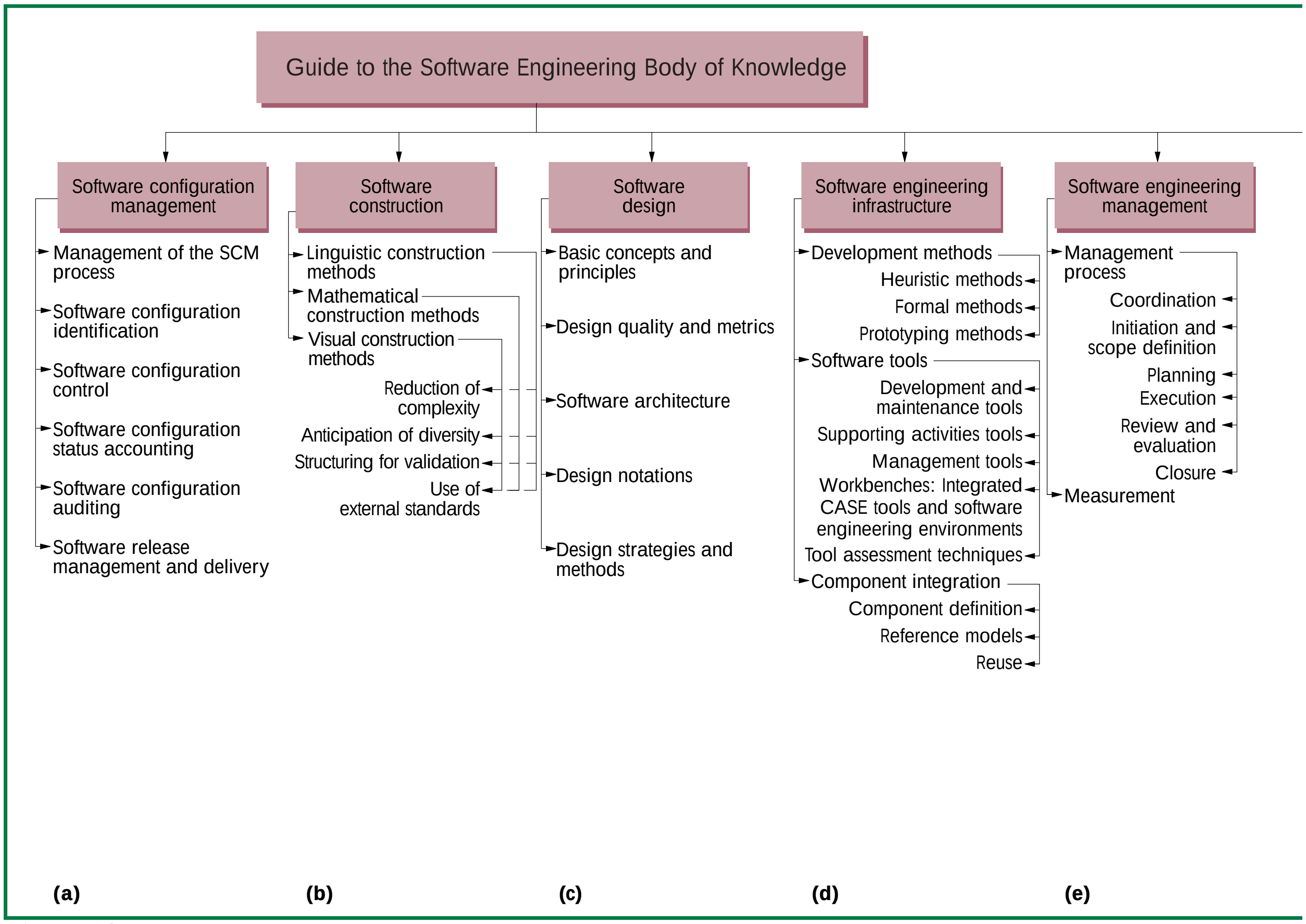
To provide an alternative manner for viewing the topics and connecting to other engineering disciplines, the Guide will classify the topics according to the taxonomy of engineering design knowledge that Walter Vincenti proposed in his 1990 history of aeronautical engineering.[4] The six categories of engineering design knowledge are fundamental design concepts, criteria and specifications, theoretical tools, quantitative data, practical considerations, and approaches to problem solving.

Ratings

As an aid, notably to curriculum developers, the Guide will also rate each topic with a set of pedagogical categories commonly attributed to Benjamin Bloom. The concept is that educational objectives can be classified into six categories representing increasing depth: knowledge, comprehension, application, analysis, synthesis, and evaluation (for Bloom's taxonomy, visit http://www.valdosta.peachnet.edu/~whuitt/psy702/cogsys/bloom.html).

KAs from related disciplines

Each SWEBOK KA description will also identify relevant KAs from related disciplines. Although these KAs will be merely identified without additional description or references, they should aid curriculum developers.



Guide to the Software Engineering Body of Knowledge
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 construction
Linguistic construction methods
Mathematical construction methods
Visual construction methods
Reduction of complexity
Anticipation of diversity
Structuring for validation
Use of external standards
Software design
Basic concepts and principles
Design quality and metrics
Software architecture
Design notations
Design strategies and methods
Software engineering infrastructure
Development methods
Heuristic methods
Formal methods
Prototyping methods
Software tools
Development and maintenance tools
Supporting activities tools
Management tools
Workbenches: Integrated CASE tools and software engineering environments
Tool assessment techniques
Component integration
Component definition
Reference models
Reuse
Software engineering management
Management process
Coordination
Initiation and scope definition
Planning
Execution
Review and evaluation
Closure
Measurement
(a)
(b)
(c)
(d)
(e)

Figure 2. A mapping of the Guide to the Software Engineering Body of Knowledge.

The Knowledge Areas

The selection, titling, and descriptions of each KA remain the subject of comment, review, and amendment. Furthermore, some themes—such as measurement, tools, and standards—cut across the KAs and are currently treated separately in each. These decisions will all be reviewed in subsequent versions of the Guide. Here, in alphabetical order, we describe the KAs as currently drafted. Figure 2 maps out the 10 KAs and the important topics incorporated within them.

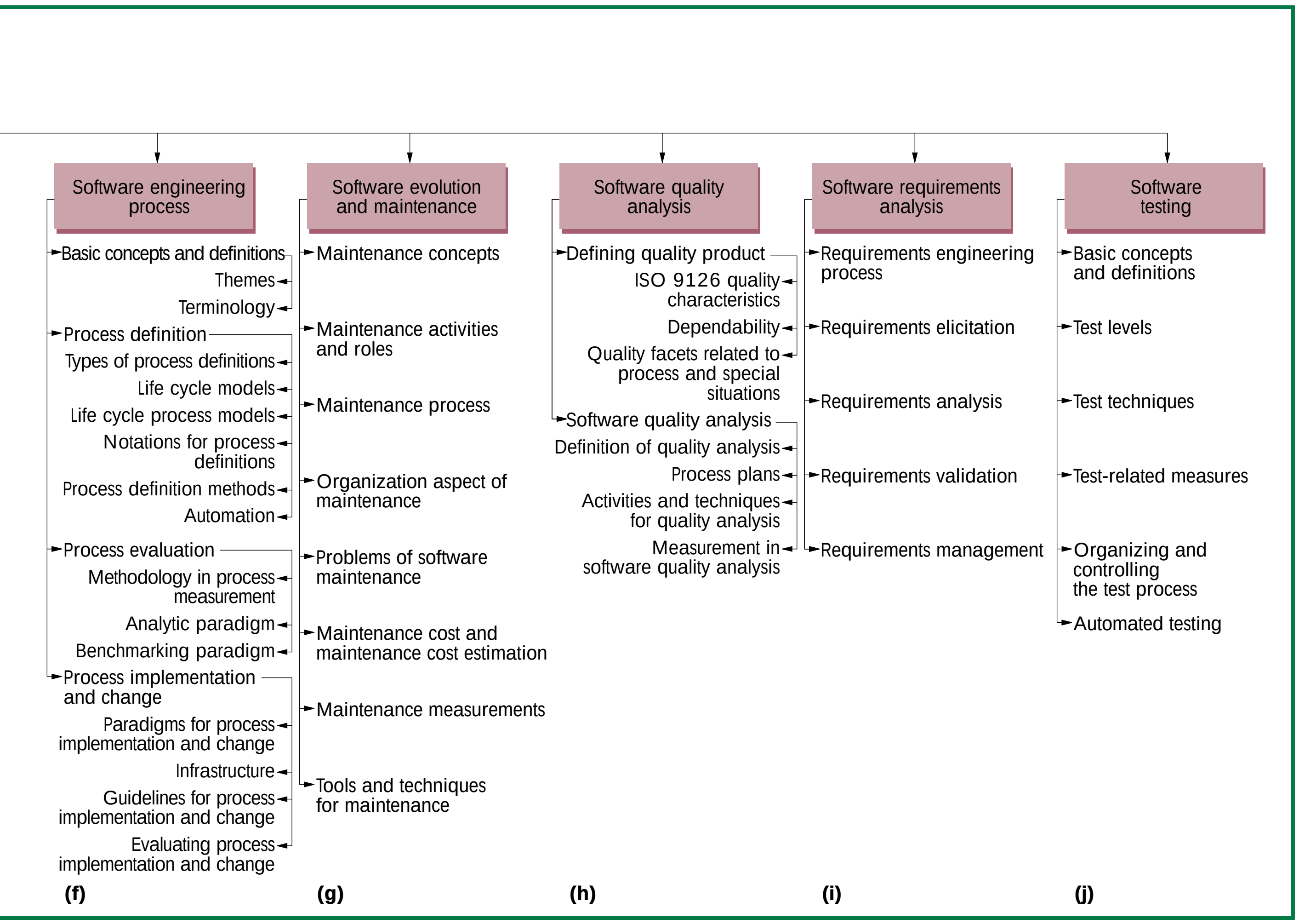
Software configuration management

We can define a system as a collection of components organized to accomplish a specific function or set of functions. A system's configuration is the function or physical characteristics of hardware, firmware, software, or a combination thereof as set forth in technical documentation and achieved in a product. Configuration management, then, is the discipline of identifying the configuration at discrete points in time to systematically control its changes and to maintain its integrity and traceability throughout the system life cycle.

The concepts of configuration management apply to all items requiring control, though there are differences in implementation between hardware configuration management and software configuration management. The primary activities of software configuration management are used as the framework for organizing and describing the topics of this KA. These primary activities are the management of the software configuration management process; software configuration identification, control, status accounting, and auditing; and software release management and delivery (see Figure 2a).

Software construction

Software construction is a fundamental act of software engineering; programmers must construct working, meaningful software through coding, self-validation, and self-testing (unit testing). Far from


Software engineering process
Basic concepts and definitions
Themes
Terminology
Process definition
Types of process definitions
Life cycle models
Life cycle process models
Notations for process definitions
Process definition methods
Automation
Process evaluation
Methodology in process measurement
Analytic paradigm
Benchmarking paradigm
Process implementation and change
Paradigms for process implementation and change
Infrastructure
Guidelines for process implementation and change
Evaluating process implementation and change
(f)
Software evolution and maintenance
Maintenance concepts
Maintenance activities and roles
Maintenance process
Organization aspect of maintenance
Problems of software maintenance
Maintenance cost and maintenance cost estimation
Maintenance measurements
Tools and techniques for maintenance
(g)
Software quality analysis
Defining quality product
ISO 9126 quality characteristics
Dependability
Quality facets related to process and special situations
Software quality analysis
Definition of quality analysis
Process plans
Activities and techniques for quality analysis
Measurement in software quality analysis
(h)
Software requirements analysis
Requirements engineering process
Requirements elicitation
Requirements analysis
Requirements validation
Requirements management
(i)
Software testing
Basic concepts and definitions
Test levels
Test techniques
Test-related measures
Organizing and controlling the test process
Automated testing
(j)

being a simple mechanistic translation of good design in working software, software construction burrows deeply into some of the most difficult issues of software engineering.

The breakdown of topics for this KA adopts two complementary views of software construction. The first view comprises three major styles of software construction interfaces: linguistic, mathematical, and visual (see Figure 2b). For each style, topics are listed according to four basic principles of organization that strongly affect the way software construction is performed: reducing complexity, anticipating diversity, structuring for validation, and using external standards.

For example, the topics listed under anticipation of diversity for linguistic software construction methods are information hiding, embedded documentation, complete and sufficient method sets, object-oriented class inheritance, creation of "glue" languages for linking legacy components, table-driven software, configuration files, and self-describing software and hardware.

Software design

Design transforms requirements—typically stated in terms relevant to the problem domain—into a description explaining how to solve the problem. It describes how the system is decomposed and organized into components, and it describes the interfaces between these components. Design also refines the description of these components into a level of detail suitable for initiating their construction.

Basic concepts and principles of software design constitute the first subarea of this KA (see Figure 2c). Design quality and metrics constitutes the second subarea and is divided into quality attributes, quality assurance, and metrics. Software architecture is the next subarea and includes topics on structures and viewpoints, architectural descriptions, patterns, and object-oriented frameworks. It also includes a section on architectural styles—an important notion in the field of software architecture—which presents some of the major styles various authors have identified.

The design notations subarea discusses notations for documenting a specific high-level design or for producing a detailed system design. Design strategies and methods constitute the last subarea, and it contains four main topics: general strategies, data-structure-centered design, function-oriented design, and object-oriented design.

Software engineering infrastructure

This KA covers three subareas that cut across the other KAs: development methods, software tools, and component integration (see Figure 2d).

Development methods impose structure on software development and maintenance activity with the goal of making the activity systematic and ultimately more successful. Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks, and guidelines for checking both the process and product. Development methods vary widely in scope, from a single life-cycle phase to the complete life cycle. The SWEBOK Guide will divide this subarea into three nondisjointed main topics: heuristic methods dealing with informal approaches, formal methods dealing with mathematically based approaches, and prototyping methods dealing with approaches based on various forms of prototyping.

The emergence of software components as a viable approach to software development represents a maturing of the discipline.

Software tools are the computer-based tools intended to assist the software engineering process. Tools are often designed to support particular methods, reducing the administrative load associated with applying the method manually. Like methods, they are intended to make development more systematic, and they vary in scope from supporting individual tasks to encompassing the complete life cycle. The top-level partitioning of the software tools subarea distinguishes between development and maintenance, supporting activities, and management tools. The remaining categories cover integrated tool sets (also known as software engineering environments) and tool assessment techniques.

The emergence of software components as a viable approach to software development represents a maturing of the discipline to overcome the not-invented-here syndrome. The component integration subarea is partitioned into topics dealing with individual components, reference models that describe how components can be combined, and the more general topic of reuse.

Software engineering management

The software engineering management KA consists of both the management process and measurement subareas (see Figure 2e). While these two areas are often regarded (and generally taught) as being separate, and indeed they do possess many mutually unique aspects, their close relationship motivates the combined treatment the Guide adopts. In essence, management without measurement—qualitative or quantitative—suggests a lack of rigor, and measurement without management suggests a lack of purpose or context.

The management process subarea considers the notion of management "in the large" under the coordination topic, addressing issues such as project selection, standards development and implementation, project staffing, and team development. It organizes the remaining topics according to stages in the project development life cycle: initiation and scope definition, planning (including schedule and cost estimation and risk assessment), execution, review and evaluation, and closure.

The measurement subarea addresses four topics: measurement program goals, measurement selection, data collection, and model development. The first three topics are primarily concerned with the theory and purpose behind measurement, and they address issues such as measurement scales and measure selection. Another issue included is the collection of measures, which involves both technical issues (automated extraction) and human issues (questionnaire design and responses to measurements being taken). The fourth topic (model development) is concerned with using both data and knowledge to build models.

Software engineering process

This KA covers the definition, implementation, measurement, management, change, and improvement of software processes. The first subarea—basic concepts and definitions—establishes the KA themes and terminology (see Figure 2f).

The purpose and methods for defining software processes, as well as existing software process definitions and automated support, are described in the process definition subarea. The topics of this subarea are types of process definitions, life-cycle models, life-cycle process models, notations for process definitions, process definition methods, and automation.

The process evaluation subarea describes the approaches for the qualitative and quantitative analysis of software processes. Measurement plays an important role in process evaluation; therefore, methodology in process measurement is this subarea's first topic. Two general paradigms, analytic and benchmarking, distinguish between types of evaluations. The analytic paradigm relies on quantitative evidence to determine where improvements are needed and whether an improvement initiative has been successful. Under this paradigm falls qualitative evaluation, root-cause analysis, process simulation, orthogonal defect classification, experimental and observational studies, and personal software process. The benchmarking paradigm depends on identifying an excellent organization in a field and documenting its practices and tools. Process assessment models and methods are the two main topics listed under this paradigm.

The process implementation and change subarea describes the paradigms, infrastructure, and critical success factors necessary for successful process implementation and change. The topics of this subarea are paradigms for process implementation and change, infrastructure, guidelines for process implementation and change, and evaluating process implementation and change.

Software evolution and maintenance

Software maintenance is defined by *IEEE Standard 1219-1998, IEEE Standard for Software Maintenance* as modifying a software product after delivery to correct faults or improve performance or other attributes, or to adapt the product to a modified environment. However, software systems are rarely completed and constantly evolve over time. Therefore, this KA also includes topics relevant to software evolution.

The maintenance concepts subarea defines maintenance, its basic concepts, and how the concept of system evolution fits into software engineering (see Figure 2g). It also explains the duties that maintainers perform. The maintenance activities and roles subarea addresses the formal types of maintenance and common activities. As with software development, the process is critical to the success and understanding of software evolution and maintenance. The next subarea discusses standard maintenance processes. Organizing maintenance might differ from development; the subarea on organizational aspects discusses the differences.

Software evolution and maintenance present unique and different technical and managerial problems for software engineering, as addressed in the problems of software maintenance subarea. Cost is always a critical topic when discussing software evolution and maintenance. The subarea on maintenance cost and maintenance cost estimation concerns life-cycle costs as well as costs for individual evolution and maintenance tasks. The maintenance measurements subarea addresses the topics of quality and metrics. The final subarea, tools and techniques for maintenance, aggregates many subtopics that the KA description otherwise fails to address.

Software quality analysis

Production of quality products is the key to customer satisfaction. Software without the requisite features and degree of quality is an indicator of failed (or at least flawed) software engineering. However, even with the best software engineering processes, requirement specifications can miss customer needs, code can fail to fulfill requirements, and subtle errors can lie undetected until they cause minor or major problems—even catastrophic failures. This KA therefore discusses the knowledge related to software quality assurance and software verification and validation activities.

The goal of software engineering is a quality product, but quality itself can mean different things. Despite different terminology, there is some consensus about the attributes that define software quality and dependability over a range of products.

Even with the best software engineering processes, requirement specifications can miss customer needs.

These definitions provide the base knowledge from which individual quality products are planned, built, analyzed, measured, and improved. The defining quality products subarea discusses these definitions (see Figure 2h).

Software quality assurance is a process designed to assure a quality product; it is a planned and systematic pattern of all actions necessary to provide adequate confidence that the product conforms to specified technical requirements. Software verification and validation is a process that provides an objective assessment of software products and processes throughout the software life cycle; that is, the verification and validation process lets management see into the product's quality.

These two processes form the backbone of the software quality analysis subarea, which is divided into four main topics: definition of quality analysis, process plans, activities and techniques for quality analysis, and measurement in software quality analysis.

Software requirements analysis

The software requirements analysis KA is broken down into five subareas that correspond approximately to process tasks that are often enacted concurrently and iteratively rather than sequentially (see Figure 2i).

The requirements engineering process subarea introduces the requirements engineering process, orients the remaining four subareas, and shows how requirements engineering dovetails with the overall software life cycle. This section also deals with contractual and project organization issues.

The requirements elicitation subarea covers what is sometimes termed requirements capture, discovery, or acquisition. It is concerned with where requirements come from and how they can be collected by the software engineer. Requirements elicitation is the first stage in building an understanding of the problem the software must solve. It is fundamentally a human activity, and it identifies the stakeholders and establishes relationships between the development team and customer.

The requirements analysis subarea is concerned with the process of analyzing requirements to detect and resolve conflicts between them, to discover the boundaries of the system and how it must interact with its environment, and to elaborate user requirements to software requirements.

The requirements validation subarea checks for omissions, conflicts, and ambiguities and ensures that the requirements follow prescribed quality standards. The requirements should be necessary, sufficient, and described in a way that leaves as little room as possible for misinterpretation.

The requirements management subarea spans the whole software life cycle. It is fundamentally about change management and maintaining the requirements in a state that accurately mirrors the software that will—or that has been—built.

Software testing

Software testing consists of dynamically verifying a program's behavior on a finite set of test cases—suitably selected from the usually infinite domain of executions—against the specified expected behavior. These and other basic concepts and definitions constitute this KA's first subarea (see Figure 2j).

This KA divides the test levels subarea into two orthogonal breakdowns, the first of which is organized according to the traditional phases for testing large software systems . The second breakdown concerns testing for specific conditions or properties.

The next subarea describes the knowledge relevant to several generally accepted test techniques. It classifies these techniques as being either specification-based, code-based, fault-based, usage-based, or specialized. The KA deals with test-related measures in their own subarea. The next subarea expands on issues relative to organizing and controlling the test process, including management concerns and test activities. The automated testing subarea addresses existing tools and concepts related to automating the test process.

The Project

Since 1993, the IEEE Computer Society and the ACM have cooperated in promoting the professionalization of software engineering through their joint Software Engineering Coordinating Committee (SWECC) (visit http://www.computer.org/tab/swecc).

The SWEBOK project's scope, the variety of communities involved, and the need for broad participation require full-time rather than volunteer management. For this purpose, the SWECC contracted the Software Engineering Management Research Laboratory at the University of Quebec at Montreal to manage the effort. It operates under SWECC supervision.

The project team developed two important principles for guiding the project: *transparency* and *consensus*. By transparency, we mean that the development process is itself documented, published, and publicized so that important decisions and status are visible to all concerned parties. By consensus, we mean that the only practical method for legitimizing a statement of this kind is through broad participation and agreement by all significant sectors of the relevant community. By the time we complete the Stoneman version of the Guide, literally hundreds of contributors and reviewers will have touched the product in some manner.

Project contributors

Like any software project, the SWEBOK project has many stakeholders—some of which are formally represented. An Industrial Advisory Board, composed of

How to Contribute to the Project

Those interested in participating in the third review cycle to be conducted on the entire Guide to the Software Engineering Body of Knowledge can volunteer by signing up at the project's Web site, http://www.swebok.org. The transition from Stoneman to Ironman will be based primarily on feedback received from trial applications of the Stoneman guide. Those interested in performing trial applications (contact Pierre Bourque at bourque.pierre@uqam.ca).

representatives from industry (Boeing, the National Institute of Standards and Technology, the National Research Council of Canada, Raytheon, and SAP Labs-Canada) and professional societies (the IEEE Computer Society and ACM), provides financial support for the project. The IAB's generous support permits us to make the products of the SWEBOK project publicly available without any charge (visit http://www.swebok.org). IAB membership is supplemented with related standards bodies (IEEE Software Engineering Standards Committee and ISO/IEC JTC1/SC7) and related projects (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.

We realize, however, that an implicit body of knowledge already exists in textbooks on software engineering. Thus, to ensure we correctly characterize the discipline, Steve McConnell, Roger Pressman, and Ian Sommerville—the authors of three best-selling textbooks on software engineering—have agreed to serve on a Panel of Experts, acting as a voice of experience. In addition, the extensive review process (described later) involves feedback from relevant communities. In all cases, we seek international participation.

Normative literature

The project differs from previous efforts in its relationship to normative literature. Most of the 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 rather than describing the variety of things that the engineer might or can do. 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 are represented in the project. In fact, a preliminary outline of KAs was based directly on the 17 processes described in *ISO/IEC 12207, Software Life Cycle Processes*. Ultimately, we hope that software engineering practice standards will contain principles traceable to the SWEBOK Guide.

Reviews

We organized the development of the Stoneman version into three public review cycles. The first review cycle focused on the soundness and reasonableness of the proposed breakdown of topics within each KA. Thirty-four domain experts completed this review cycle in April 1999. The reviewer comments, as well as the identities of the reviewers, are available on the project's Web site.

The second review cycle was organized around the guidelines we originally gave to the KA specialists. A considerably larger group of professionals, organized into review viewpoints, answered a detailed questionnaire for each KA description. The viewpoints (for example, individual practitioners, educators, and makers of public policy) were formulated to ensure relevance to the Guide's various intended audiences. The reviewer feedback collected in this review cycle, completed in October 1999, is also available on the project's Web site. KA specialists will document how reviewer feedback was resolved in the KA descriptions.

The focus of the third review cycle will be on the correctness and utility of the Guide. This review cycle, currently scheduled to start in January 2000, will be completed by individuals and organizations representing a cross section of potential interest groups (see the "How to contribute to the project" sidebar). We've already recruited hundreds of professionals to review the entire Guide, and we're soliciting more to fulfill our coverage objectives.

Throughout the project, the SWEBOK team has ensured that there is always material available to tangibly capture the project's progress. Most of this material is available publicly on the project's Web site. (Out of courtesy to the KA specialists, draft material is withheld until completed.) The project team is currently updating the KA descriptions based on the results of the second review cycle. Early in 2000, we'll invite major professional societies and the software engineering community to participate in the third review cycle and comment on the entire Guide. The completed Stoneman guide will then be made available on the Web in Spring 2000, and, to the extent possible, the cited reference materials will also be made freely available.

Prior to developing the Ironman version of the Guide, we'll use the Stoneman guide in experimental application to provide feedback on its usability. Although the extent of coverage is intended to be identical, developing Ironman will involve a broader, more exhaustive review process, based on feedback from trial usage of the Guide. ❖

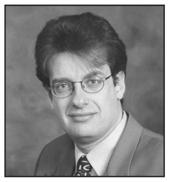
Acknowledgments

The SWEBOK project team gratefully acknowledges the support provided by the members of the Industrial Advisory Board. Funding for this project is provided by the Association for Computing Machinery, Boeing, the IEEE Computer Society, the National Institute of Standards and Technology, the National Research Council of Canada, Raytheon, and SAP Labs (Canada). The team also appreciates the important work performed by the KA specialists named in the article. Finally, the team acknowledges the indispensable contribution of the reviewers who have participated so far.

References

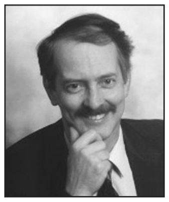
1. P. Bourque et al., *Guide to the Software Engineering Body of Knowledge: A Strawman Version*, University of Quebec at Montreal, 1998; http://www.swebok.org (current Oct. 1999).
2. D.J. Bagert et al., *Guidelines for Software Engineering Education, Version 1.0*, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, Nov. 1999; http://www.sei.cmu.edu/collaborating/ed/workgroup-ed.html (current Oct. 1999).
3. Project Management Institute, *A Guide to the Project Management Body of Knowledge*, Upper Darby, Pa., 1996; http://www.pmi.org/publictn/pmboktoc.htm (current Oct. 1999).
4. W. Vincenti, *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*, The Johns Hopkins Univ. Press, Baltimore, 1990.

About the Authors



Pierre Bourque is the director of applied research at the Software Engineering Management Research Laboratory at the University of Quebec at Montreal. He is also a coeditor of the SWEBOK project. He has published and spoken internationally on software measurement, functional size measurement, project duration modeling, fundamental principles of software engineering, software reengineering, IT governance, and software engineering standards. He received his MSc in mathematics (computer science) from the Université de Sherbrooke. Contact him at bourque.pierre@uqam.ca.

Robert Dupuis is a professor and the director of four graduate programs, including the MSc program in software engineering at the University of Quebec at Montreal. He is also a coeditor of the SWEBOK project. His teaching activities have included software engineering, technology assessment, research methodology, computer ethics, and the diffusion of technology. His main research interests include software engineering, the diffusion of end-user computing, and the diffusion and evaluation of legal expert systems. He received his PhD in management sciences from the Université de Montpellier II, France. Contact him at dupuis.robert@uqam.ca.



Alain Abran is a professor and the director of the Software Engineering Management Research Laboratory at the University of Quebec at Montreal. He is also the coexecutive editor of the SWEBOK project. His research interests include functional size measurement, software productivity and estimation models, risk management, and software quality. He holds an MS in management sciences and an MS in electrical engineering, both from the University of Ottawa, and a PhD in software engineering from the École Polytechnique de Montréal. He is actively involved in international software engineering standards and cochairs the Common Software Measurement International Consortium. Contact him at abran.alain@uqam.ca.

James W. Moore's biography appears in his article on page 57.

Leonard Tripp's biography appears in the Guest Editors' Introduction on page 18.

Readers can contact the authors in care of James W. Moore at The MITRE Corp., 1820 Dolley Madison Blvd., W534, McLean, VA 22102; james.w.moore@ieee.org.