The Emerging Consensus on the Software Engineering Body of Knowledge

Pierre Bourque, Robert Dupuis, and Alain Abran
Université du Québec à Montréal
C.P. 8888, Succ. Centre-Ville
Montréal, Québec, Canada H3C 3P8
Email: {bourque.pierre, dupuis.robert, abran.alain}@uqam.ca

James W. Moore
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, Virginia 22102-3481 USA
Email: James.W.Moore@ieee.org

Leonard Tripp
The Boeing Company
MS 19-MM
7701, 14 Avenue South
Seattle, WA 98108 USA
Email: l.tripp@computer.org

Abstract

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.

At the time of writing this paper in September 2000, the SWEBOK project (http://www.swebok.org) is nearing the end of the second of its three phases. Here we summarize the results to date and provide an overview of the project.

Keywords

Software engineering, software engineering profession, body of knowledge.

Objectives and Audience

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

1. Characterize the contents of the software engineering discipline.
2. Provide a 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 and licensing 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; the goal is to gain consensus on the core subset of knowledge characterizing the software engineering discipline.

To achieve these goals, the project is oriented 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 will benefit from SWEBOK, as well as the students learning the software engineering profession.

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 has organized the body of knowledge into several Knowledge Areas. The Stoneman version of the Guide identifies 10 KAs (see Table 1). In addition, we're considering seven related disciplines (see Table 2).

Table 1. The SWEBOK Knowledge Areas and their corresponding specialists.

Knowledge Area	Specialists
Software requirements	Pete Sawyer and Gerald Kotonya, Lancaster University, UK
Software design	Guy Tremblay, Université du Québec à Montréal, Canada
Software construction	Terry Bollinger, The MITRE Corporation, US, Philippe Gabrini and Louis Martin, Université du Québec à Montréal, Canada
Software testing	Antionia Bertolino, Consiglio Nazionale delle Ricerche, Italy
Software maintenance	Thomas M. Pigoski, TECHSOFT, US
Software configuration management	John A. Scott and David Nisse, Lawrence Livermore Laboratory, US
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 engineering tools and methods	David Carrington, The University of Queensland, Australia
Software quality	Dolores Wallace and Larry Reeker, National Institute of Standards and Technology, US

Table 2. Related disciplines.

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

The distinction between KAs and related disciplines is important to the Guide's purpose. The project specifies 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 does 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[1], or the Working Group on Software Engineering Education [2]. As the following sections explain, each KA description—which is around 15 pages—contains several important components.

Hierarchical organization

The Guide uses 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 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 should 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 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, the project adopted a concept of *generally accepted* knowledge, which we 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 you should apply the designated knowledge uniformly to all software

[1] See http://www.computer.org/tab/swecc/

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

Ratings

As an aid, notably to curriculum developers, the Guide also rates each topic with one of a set of pedagogical categories commonly attributed to Benjamin Bloom [3]. The concept is that educational objectives can be classified into six categories representing increasing depth: knowledge, comprehension, application, analysis, synthesis, and evaluation.

KAs from related disciplines

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

The Knowledge Areas

The selection, titling, and descriptions of each Knowledge Area remains the subject of comment, review, and amendment. These decisions will all be reviewed in subsequent versions of the Guide. Here, we describe the KAs as currently drafted at the time of writing this paper in February 2000, therefore prior to the final review of the Stoneman version. Table 1 identifies the KA specialists responsible for preparing the full KA descriptions, and Figure 1 maps out the 10 KAs and the important topics incorporated within them.

Software requirements

The software requirements KA is concerned with the acquisition, analysis specification and management of software requirements. It is broken down into six subareas that correspond approximately to process tasks that are often enacted concurrently and iteratively rather than sequentially (see Figure 1a).

The requirements engineering process subarea introduces the requirements engineering process, orients the remaining five subareas, and shows how requirements engineering dovetails with the overall software engineering process.

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 requirements 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; the requirements analysis subarea also discusses the elaboration from system requirements to software requirements. The software requirements specification subarea is concerned with the structure, quality and verification of the requirements document.

The requirements validation subarea is concerned with checking for omissions, conflicts, and ambiguities and with ensuring 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 to be—or that has been—built.

Software design

Design transforms (software) requirements—typically stated in terms relevant to the problem domain—into a description explaining how to solve the software-related aspects of 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 allowing their construction.

Basic concepts of software design constitute the first subarea of this KA (see Figure 1b). Software architecture is the next subarea and includes topics on structures and viewpoints, architectural styles and patterns, design patterns and families of programs and frameworks. Design quality analysis and evaluation constitute the next subarea and is divided into quality attributes, quality analysis and evaluation tools, and metrics.

The design notations subarea discusses notations for structural and behavioral descriptions. Design strategies and methods constitute the last subarea, and it contains four main topics: general strategies, function-oriented design, object-oriented design, data-structure-centered design and other methods.

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 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, formal, and visual (see Figure 1c). 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 interfaces are information hiding, embedded documentation (commenting), 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 (e.g., plug and play).

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 the first subarea of this KA (see Figure 1d).

This next subarea divides the test levels into the object of the test and the objectives of testing.

Then the next subarea presents various decompositions of test techniques and related criteria such as specification-based, code-based, fault-based, usage-based.

The next subarea describes the knowledge relevant to several generally accepted test techniques. It classifies these techniques as being intuition-based, specification-based, code-based, fault-based, usage-based, or based on the nature of the application. An alternative breakdown of test techniques as being white-box or black-box is also presented. Test-related measures are dealt with in their own subarea.

The next subarea expands on issues relative to the management of the test process, including management concerns and test activities.

Software maintenance

Software maintenance (see Figure 1e) 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 introduction to software maintenance subarea discusses the need for software maintenance and the categories of maintenance. The maintenance activities subarea addresses the unique and supporting activities of maintenance as well as maintenance planning. As with software development, process is critical to the success and understanding of software maintenance. The next subarea discusses standard maintenance processes. Organizing the maintenance area might differ from development; the subarea on organizational aspects discusses the differences.

Software 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 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, techniques for maintenance, aggregates many subtopics that the KA description otherwise fails to address.

Software configuration management

We can define a system as a collection of components organized to accomplish a specific function and/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 distinct 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 1f).

Software engineering management

The software engineering management KA addresses the management of software development projects and the measurement and modeling of such projects (see Figure 1g). It consists of eight subareas, from measurement, to organizational management and coordination and then to six additional subareas organized by lifecycle phases. The measurement subarea addresses four main topics: measurement program goals, measuring software and its development, measurement selection, data collection, and model development.

The organizational management and coordination subarea considers the notion of portfolio management, acquisition decisions and management, policy management, personnel management and communications. The remaining subareas are organized according to stages in the project development life cycle: initiation and scope definition, planning, enactment, review and evaluation, project close out and post-closure activities.

An alternative classification of these topics is also proposed in the KA description based on common themes.

Software engineering process

This KA covers the basic concepts and definitions, infrastructure, measurement, process definition, qualitative process analysis and process implementation and change (see Figure 1h). The first subarea—basic concepts and definitions—establishes the KA themes and terminology. The next subarea deals with process infrastructure including the experience factory and the software engineering process group.

The process measurement subarea describes the methodologies in process measurement and process measurement paradigms.

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 qualitative process analysis subarea discusses qualitative techniques to analyze software processes, to identify strengths and weaknesses.

The topics of the process implementation and change subarea are paradigms for process implementation and change, guidelines for process implementation and change, and evaluating the outcome of process implementation and change.

Software engineering tools and methods

This KA covers two themes that cut across the other KAs: software tools and software development methods (see Figure 1i).

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, management tools and infrastructure tools.

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 Guide divides this subarea into three non disjointed 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.

Software quality

Production of quality products is 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 of 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. The first subarea, software quality concepts, discusses measuring the value of quality, and other characteristics such as the quality attributes for the engineering process (see Figure 1j).

The software quality assurance process provides assurance that the software products and processes in the project life cycle conform to their specified requirements and adhere to their established plans. The software verification and validation process determines whether products of a given development or maintenance activity conform to the requirements of that activity and those imposed by previous activities, and whether the final software product (through its evolution) satisfies its intended use and user needs. These form three additional subareas.

The last subarea discusses measurement as applied to software quality assurance and verification and validation.

The Project

From 1993 to 2000, the IEEE Computer Society and the ACM have cooperated in promoting the professionalization of software engineering through their joint Software Engineering Coordinating Committee (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 Université du Québec à Montréal 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 the Stoneman version of the Guide is completed, 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 representatives from industry (Boeing, National Institute of Standards and Technology, National Research Council of Canada, Rational Software Corp., Raytheon Systems, and SAP Labs-Canada) and professional societies (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 the 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 to maintain a broad scope of relevance.

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 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.

In the second review cycle 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 results of this review cycle, completed in October 1999, are also available on the project's Web site. The focus of the third review cycle is on the correctness and utility of the Guide. The third review cycle started in the Spring of 2000 by individuals and organizations representing a cross-section of potential interest groups. The project team is currently handling the disposition of comments.

The ISO sub-committee on software engineering (ISO/IEC JTC1/SC7) has also approved the inclusion of the SWEBOK Guide into its program of work, for publication as an ISO technical report. Comments received from national standards bodies will be handled with the support of the SWEBOK project team.

Throughout the project, the SWEBOK team has ensured that there was always material available to tangibly capture the project's progress. Most of this material is available publicly on the project's Web site.

Prior to developing the Ironman version of the Guide, we will use the Stoneman guide in experimental application to provide feedback on its usability. Those interested in performing experimental applications of the Guide are invited to contact the project team.

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, Rational Software Corp., 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 hundreds of reviewers who have participated so far.

References

1. P. Bourque et al., *Guide to the Software Engineering Body of Knowledge: A Strawman Version*, Université du Québec à Montréal, 1998 (Available from www.swebok.org).
2. Bagert, D.J., Hilburn, T.B., Hislop, G., Lutz, McCracken, M., and Mengel, S., *Guidelines for Software Engineering Education*, Version 1.0 (CMU/SEI-99-TR-032, ESC-TR-99-002). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, November 1999;

http://www.sei.cum.edu/collaborating/ed/workgroup-ed-html.

3. Bloom et al., Taxonomy of the Cognitive Domain. See http://www.valdosta.peachnet.edu/~whuitt/psy702/cogsys/bloom.html

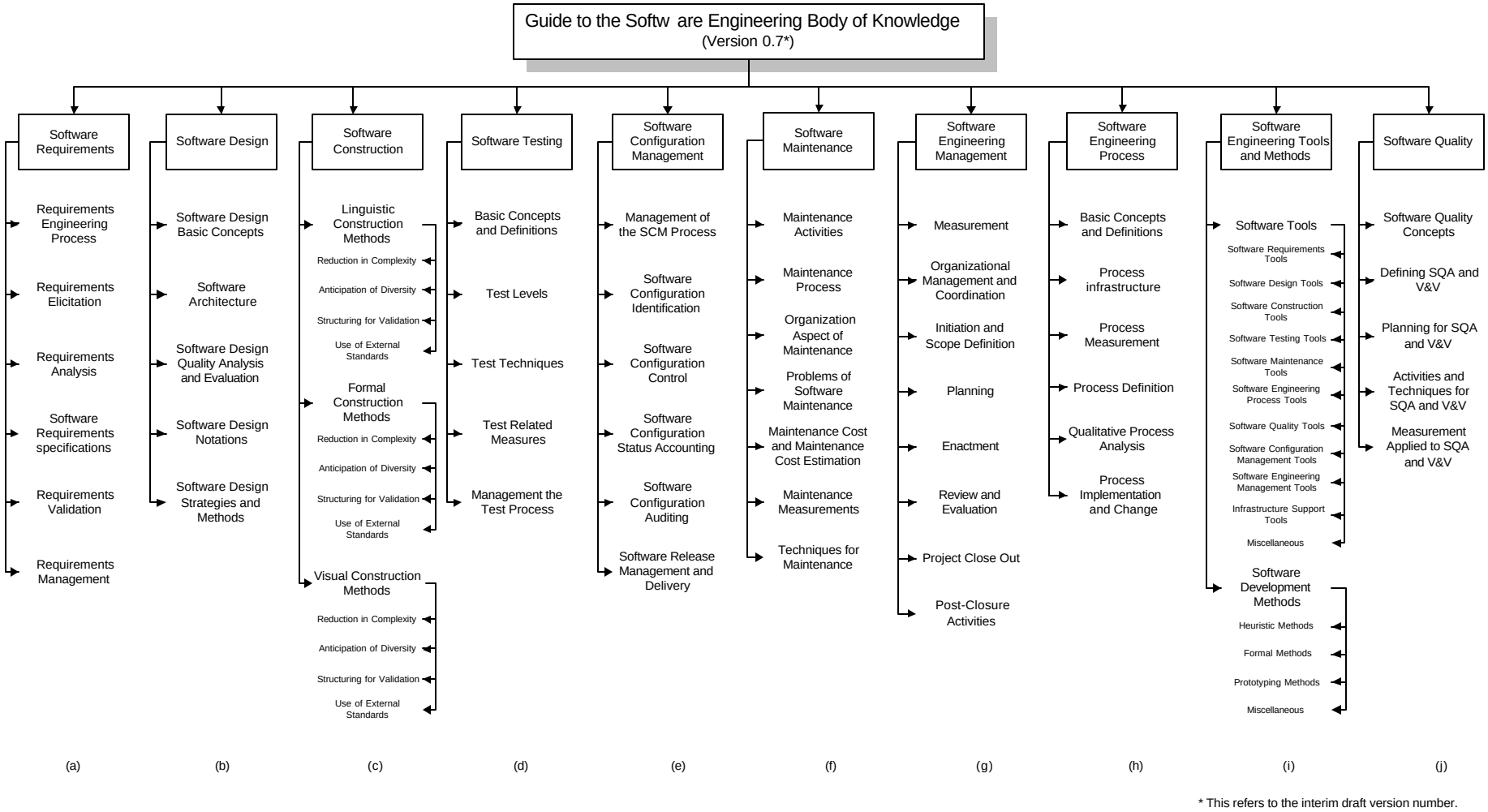

Figure 1. The taxonomy of the Guide to the Software Engineering Body of Knowledge.