CHAPTER 1

INTRODUCTION TO THE GUIDE

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.

Originally formed in 1993 by the IEEE Computer Society and the Association for Computing Machinery, the Software Engineering Coordinating Committee (SWECC) has been actively promoting software engineering as a profession and an engineering discipline.

Achieving consensus by the profession on a core body of knowledge is a key milestone in all disciplines and has been identified by the SWECC as crucial for the evolution of software engineering toward a professional status. This Guide, written under the auspices of this committee, is the part of a multi-year project designed to reach this consensus.

What is Software Engineering?

The IEEE Computer Society defines software engineering as

"(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1)."[1]

What is a Recognized Profession?

For software engineering to be known as a legitimate engineering discipline and a recognized profession, consensus on a core body of knowledge is imperative. This fact is well illustrated by Starr when he defines what can be considered a legitimate discipline and a recognized profession. In his Pulitzer-prize-winning book on the history of the medical profession in the USA, he 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."[2]

What are the Characteristics of a Profession ?

But what are the characteristics of a profession? Gary Ford and Norman Gibbs studied several recognized professions including medicine, law, engineering and accounting[3]. 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.*

This Guide contributes to 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.

What are the Objectives of the SWEBOK Project?

The 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

[1] "IEEE Standard Glossary of Software Engineering Terminology," IEEE, Piscataway, NJ std 610.12-1990, 1990.

[2] P. Starr, The Social Transformation of American Medicine: Basic Books, 1982. p. 15.

[3] G. Ford and N. E. Gibbs, "*A Mature Profession of Software Engineering*," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical CMU/SEI-96-TR-004, January 1996.

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.

The first of these objectives, the consistent worldwide view of software engineering was supported by a development process that has engaged approximately 500 reviewers from 42 countries. (More information regarding the development process can be found in the Preface and on the Web site. Professional and learned societies and public agencies involved in software engineering were officially contacted, made aware of this project and invited to participate in the review process. Knowledge Area Specialists or chapter authors were recruited from North America, the Pacific Rim and Europe. Presentations on the project were made to various international venues and more are scheduled for the upcoming year.

The second of the objectives, the desire to set a boundary, motivates the fundamental organization of the Guide. The material that is recognized as being within software engineering is organized into the ten Knowledge Areas listed in Table 1. Each of the ten KAs is treated as a chapter in this Guide. Table 1. The SWEBOK knowledge areas (KA).

Software requirements

Software design

Software construction

Software testing

Software maintenance

Software configuration management

Software engineering management

Software engineering process

Software engineering tools and methods

Software quality

In establishing a boundary, it is also important to identify what disciplines share a boundary and often a common intersection with software engineering. To this end, the guide also recognizes seven related disciplines, listed in Table 2 (See also Appendix B). Software engineers should of course know material from these fields (and the KA descriptions may make references to the fields). It is not however an objective of the SWEBOK Guide to characterize the knowledge of the related disciplines but rather what is viewed as specific to software engineering.

Table 2 Related disciplines.

Cognitive sciences and human factors

Computer engineering

Computer science

Management and management science

Mathematics

Project management

Systems engineering

Hierarchical Organization

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 detailed specifications provided by the project's editorial team to the Knowledge Area Specialists regarding the contents of the Knowledge Area Descriptions can be found in Appendix A.

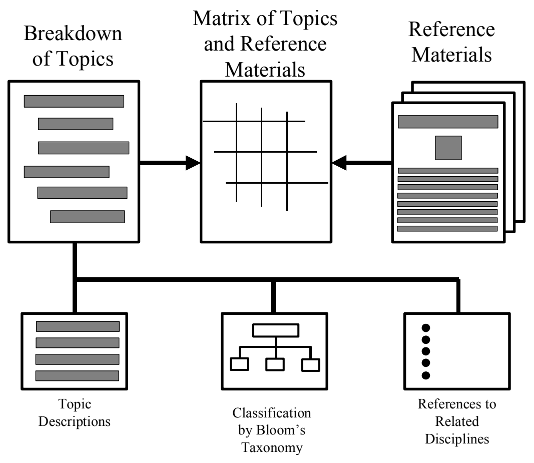

Breakdown of Topics
Matrix of Topics and Reference Materials
Reference Materials
Topic Descriptions
Classification by Bloom's Taxonomy
References to Related Disciplines

Figure 1 The organization of a KA description

The Guide uses a hierarchical organization to decompose each KA 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.

Reference Materials and a Matrix

To provide a topical access to the Knowledge—the fourth of the project's objectives—the Guide identifies reference materials for each KA including book chapters, refereed papers, or other well-recognized sources of authoritative information[4]. Each KA description also includes a matrix that relates the 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 does not attempt to be comprehensive in its citations. Much material that is both suitable and excellent is not referenced. Materials were selected, in part, because— taken as a collection—they provide coverage of the described topics.

Depth of Treatment

From the outset, the question arose as to the depth of treatment the Guide should provide. We adopted an approach that supports the fifth of the project's objectives—providing a foundation for curriculum development, certification and licensing. We applied a criterion of *generally accepted* knowledge, 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). A second definition of *generally accepted* comes from the Project Management Institute: "The generally accepted knowledge applies to most projects most of the time, and widespread consensus validates its value and effectiveness".[5]

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. 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 U.S. 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 KA descriptions are somewhat forward-looking—we're considering 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 Guide rates each topic with one of a set of pedagogical categories commonly attributed to Benjamin Bloom[6]. The concept is that educational objectives can be classified into six categories representing increasing depth: knowledge, comprehension, application, analysis, synthesis, and evaluation Results of this exercise for all KAs can be found in Appendix C. This Appendix must however not be viewed as a definitive classification but much more as a starting point for curriculum developers.

KAs from Related Disciplines

A list of disciplines (Related Disciplines) that share a common boundary with software engineering can be found in Appendix B. Appendix B also identifies from an authoritative source a list of KAs of these Related Disciplines.

A proposed Breakdown for an Additional KA

One of the knowledge areas that was not included in this Trial version because there was no consensus on the generally accepted set of reference material is *Component integration*. Since such a consensus may appear in the near future, we include in Appendix D a proposal for a breakdown of topics on that subject. This is intended to serve as a jumpstart for future work on the topic.

We recognize also that Human-Computer Interface is important and we will in future versions indicate a point beyond which the software engineer should seek the help of a specialist. There was also no consensus on a set of reference material on the subject.

THE KNOWLEDGE AREAS

Figure 2 maps out the 10 KAs and the important topics incorporated within them. The first five KAs are presented in traditional waterfall lifecycle sequence. The subsequent Kas are presented in alphabetical order. This is identical to the sequence in which they are presented in the Guide. Brief summaries of the KA descriptions appear next.

[4] Web pages in the Recommended References sections were verified on April 9, 2001.

[5] Project Management Institute, A Guide to the Project Management Body of Knowledge, Upper Darby, PA, 1996, http://www.pmi.org/publictn/pmboktoc.htm/. "Project" in the quote refers to projects in general.

[6] See chiron.valdosta.edu/whuitt/col/cogsys/bloom.html for a short description of Bloom's taxonomy. The original source is Bloom, B.S. (Ed.) (1956) Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain. New York ; Toronto: Longmans, Green.

SOFTWARE REQUIREMENTS (see Figure 2, column a)

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

The first knowledge sub-area is the *requirement engineering process*, which 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 sub-area is *requirements elicitation*, which is concerned with where requirements come from and how they can be collected by the requirements engineer. It includes requirement sources and techniques for elicitation.

The third sub-area, *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 and requirements allocation and requirements negotiation.

The fourth sub-area 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 sub-area 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.

SOFTWARE DESIGN (see Figure 2, column b)

According to the IEEE, software design 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. The knowledge area is divided into six sub-areas.

The first one presents the *basic concepts* and notions which form an underlying basis to the understanding of the role and scope of software design. These are general concepts, the context of software design, the design process and the enabling techniques for software design.

The second sub-area regroups the *key issues of software d*esign. They include concurrency, control and handling of events, distribution, error and exception handling, interactive systems and persistence.

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

The fourth sub-area describes *software design quality analysis and evaluation*. While a whole knowledge area is devoted to software quality, this sub-area 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 sub-area 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, like formal and transformational methods.

SOFTWARE CONSTRUCTION (see Figure 2, column c)

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

SOFTWARE TESTING (see Figure 2, column d)

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. It includes five sub-areas.

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 sub-area is the *test levels*. They are divided between the targets and the objectives of the tests.

The third sub-area are the *test techniques* themselves. A first category is grouped on the criterion of the base on which tests are generated, and a second group based on the ignorance of knowledge of implementation. A discussion of how to select and combine the appropriate techniques is presented.

The fourth sub-area 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 sub-area describes the *management* specific to the test process. It included management concerns and the test activities.

SOFTWARE MAINTENANCE (see Figure 2, column e)

Once in operation, anomalies are uncovered, operating environments change, and new user requirements surface. The maintenance phase of the lifecycle commences upon delivery but maintenance activities occur much earlier. The Software maintenance knowledge area is dived into six sub-areas.

The first on presents the domain's *basic concepts*, definitions, the main activities and 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 regroups *key issues* related to software maintenance. The topics covered are technical, management, cost and estimation and measurement issues.

Techniques for maintenance constitute the fourth sub-area. Those techniques include program comprehension, re-engineering, reverse engineering and impact analysis.

SOFTWARE CONFIGURATION MANAGEMENT (see Figure 2, column f)

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 software configuration and maintaining the integrity and traceability of the configuration throughout the system lifecycle. This Knowledge Area includes six sub-areas.

The first sub-area 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 sub-area are first 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, first, requesting, evaluating and approving software changes, and, second, implementing software changes, and third deviations and waivers.

The fourth sub-area 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 sub-area is *software release management and delivery*, covering software building and software release management.

SOFTWARE ENGINEERING MANAGEMENT (see Figure 2, column g)

Whilst 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. There are three sub-areas for software engineering management.

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

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

The third and last sub-area 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 metric models.

SOFTWARE ENGINEERING PROCESS (see Figure 2, column h)

The Software Engineering Process Knowledge Area is concerned with the definition, implementation, measurement, management, change and improvement of the software engineering process itself. It is divided into six sub-areas.

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

The second sub-area is *process infrastructure*, where the Software Engineering Process group concept is described, as well as the Experience Factory.

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

The fourth sub-area 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 sub-area presents *qualitative process analysis*, especially the process definition review and root cause analysis.

Finally, the sixth sub-area 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 (see Figure 2, column i)

The Software Engineering Tools and Methods knowledge area includes both the software development environments and the development methods knowledge areas identified in the Straw Man version of the guide.

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 Stone Man Version of the Guide to the Software Engineering Body of Knowledge. The first five subsections correspond to the five Knowledge Areas (*Requirements, Design, Construction, Testing, and Maintenance*) and the next four subsections correspond to the remaining Knowledge Areas (*Process, Quality, Configuration Management and Management*). 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 (see Figure 2, column j)

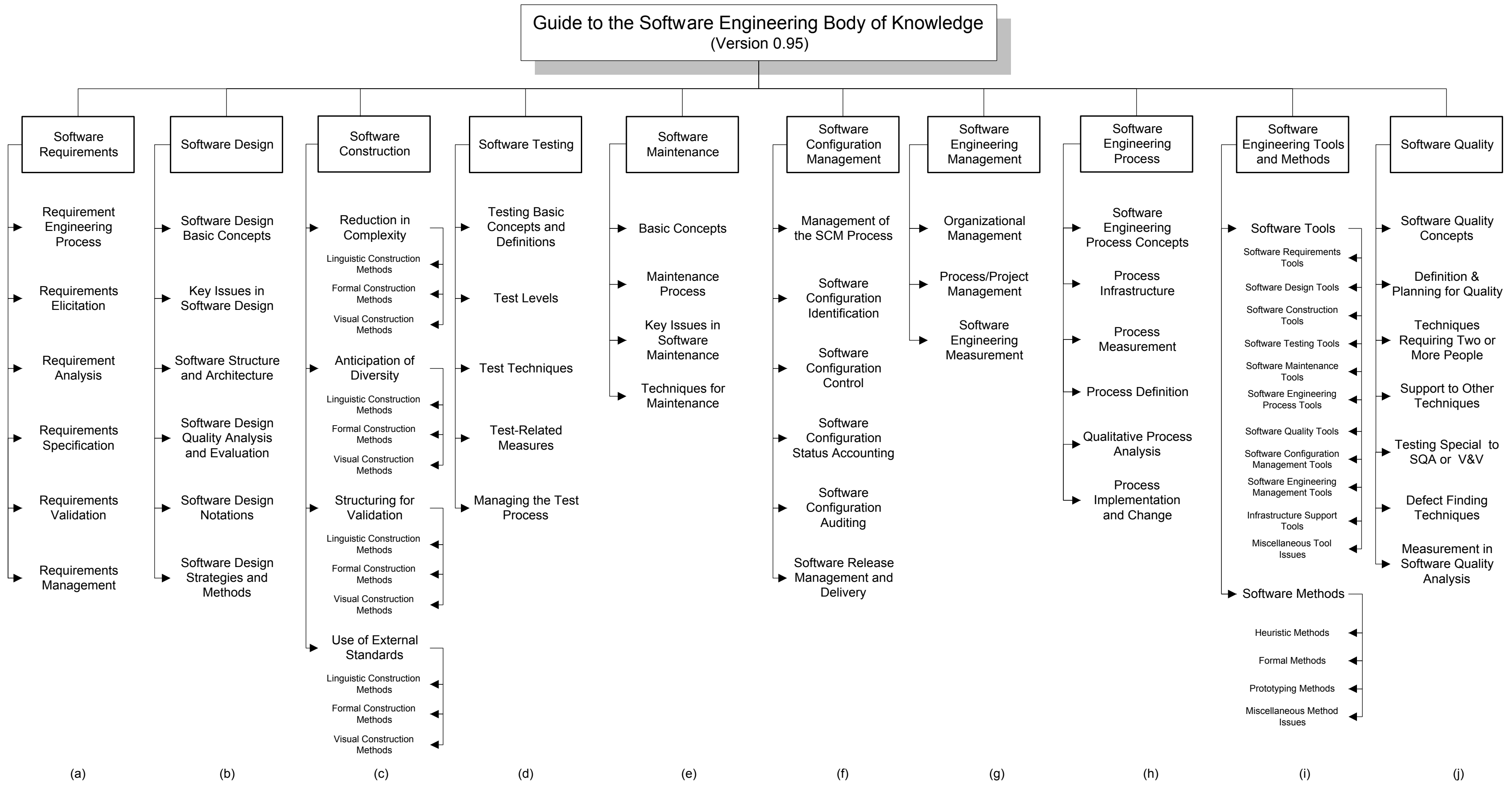
This chapter deals with software quality considerations that transcend the lifecycle processes. Since software quality is a ubiquitous concern in software engineering, it is considered in many of the other KAs and the reader will notice pointers those KAs through this KA. The Knowledge Area description covers four sub-areas.

The first sub-area describes the *software quality concepts* such as measuring the value of quality, the ISO9126 quality description, dependability and other special types of system and quality needs.

The second sub-area 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&S plans.

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

The fourth sub-area 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.


Guide to the Software Engineering Body of Knowledge
(Version 0.95)
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)