The quality concepts and subconcepts in SWEBOK: An ontology challenge

Cornelius Wille+, Alain Abran, Jean Marc Desharnais, Reiner R. Dumke+
École de Technologie Supérieure - ETS
1100 Notre-Dame Ouest, H3C 1K3 Montréal Québec , Canada,
(aabran,jmdeshar)@ele.etsmtl.ca
+Otto-von-Guericke-University of Magdeburg, Faculty of Computer Science,
Postfach 4120, 39016 Magdeburg, Germany,
(dumke,wille)@ivs.cs.uni-magdeburg.de

Abstract: *The Guide to the Software Engineering Body of Knowledge (SWEBOK) has been developed to represent an international consensus formed through broad public participation in the review process and is now close to final approval as ISO/IEC TR 19759. This guide constitutes an integrated structuring of a large set of software engineering concepts developed individually over the past forty years from a large number of distinct viewpoints. The absence of a recognized consensus on software engineering terminology has been a challenging task in building the SWEBOK Guide, and in achieving an international consensus. While major consensus has been reached at the broad taxonomy level of SWEBOK, some work remains to increase terminology consistency at a more detailed level. This paper briefly presents SWEBOK and related terminology issues. We then present the ontology approach to building domain-specific ontologies and show how it can be used to build the SWEBOK ontology and to increase its internal consistency and clarity. A specific example of the benefits of an ontology is presented, along with an analysis of the use of the term 'quality' in the current version of the SWEBOK Guide.*

Keywords: *Software Engineering Body of Knowledge, SWEBOK, ISO/IEC TR 19759, Ontology, quality, software quality*

1 SWEBOK

Articulating a Software Engineering Body of Knowledge (SWEBOK), and gaining the widest possible consensus on its content, is an essential step toward developing a profession because SWEBOK will represent broad agreement on 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 IEEE-Computer Society has championed the development of such an international consensus on a compendium and guide to the body of knowledge that has been developing and evolving over the past four

decades: the Guide to the Software Engineering Body of Knowledge (SWEBOK) project[1]
[1].

SWEBOK knowledge is subdivided into ten Knowledge Areas (KAs) – see Figure 1. To provide a topical access to the knowledge, each Knowledge Area is further broken down into topics and sub-topics, and identifies as well the related seminal reference material and a matrix linking the reference material to the topics listed. In the OO paradigm, the 10 Knowledge Areas could be considered as subclasses of the SWEBOK super class. Every concept about software engineering would be a subclass of one or more of the Knowledge Areas. That means that a concept should be a subclass of the super class and have relations to different knowledge areas. But super classes and subclasses, as well as the definitions of the concepts, represent only a first step. A SWEBOK user is not only interested in the definitions of the concepts, but also in much more detailed information about the topics that are important to him. In SWEBOK this detailed level of information is not in the Guide itself, but in its internationally approved list of references.

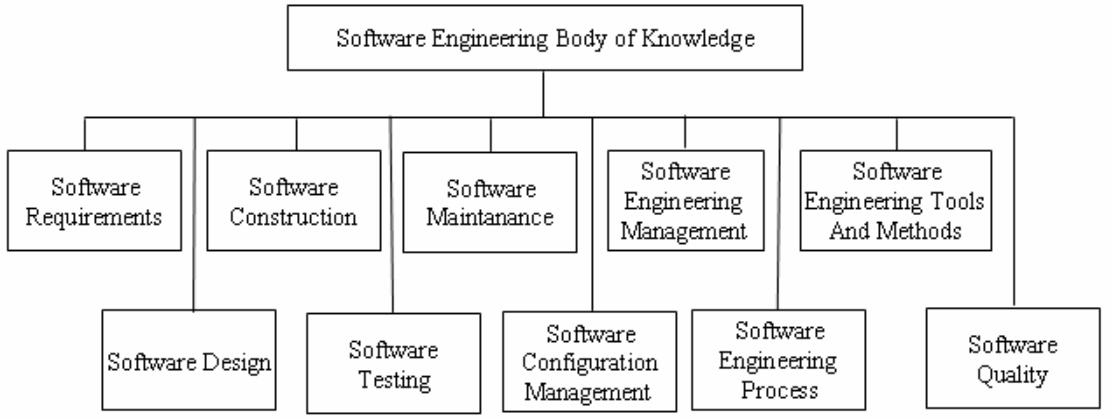

Figure 1: Knowledge Areas of the Software Engineering Body of Knowledge

The authors and hundreds of reviewers from 42 countries have contributed to SWEBOK and, in parallel, the document is being reviewed by national software engineering standardization committees with a view to becoming an internationally accepted document at the ISO level, that is, ISO/IEC TR 19759.

Because many authors have contributed to the initial versions of the SWEBOK Guide, it is necessary to verify the coherency and clarity of the terminology used within each chapter and across all chapters. For instance, in the SWEBOK

[1] The SWEBOK project has received support from the following organizations: Boeing, Raytheon, The MITRE Corporation, National Institute of Standards & Technology (USA), CONSTRUX Software, Rational Software, SAP Lab. Canada, NRC, and Canadian Council of Professional Engineers.

Guide (Trial Version 1.00), the term *quality* is used 340 times and the word *software quality* 104 times.

Terms such as 'quality', 'measurements', 'process' are used extensively in the SWEBOK Guide, but each of these terms might refer to many concepts used in different contexts and at different conceptual levels. This makes it challenging for beginner users of the Guide to recognize whether or not different subconcepts are being discussed when they are not identified as such by the use of distinct terms or expressions. It is therefore necessary to verify the precise interpretation of each of these terms throughout the text and to ensure that they are adequately identified in order to improve the understandability of the SWEBOK Guide at a detailed level.
Even though the SWEBOK Guide has already been reviewed extensively, and is going through another major review cycle during the summer of 2003, up to now no special techniques have been used to detect such terminology issues.
We illustrate this terminology issue in section 2, with an inventory of the uses of the term 'quality' in the SWEBOK guide, then in section 3 we propose a structuring of the quality knowledge embedded in this Guide. In section 4, the generic domain of ontologies is presented, and in section 5 a method of using an ontology for SWEBOK is proposed. Finally, in section 6, recommendations are given for building automated support for the construction of a full SWEBOK ontology.

2 The 'quality' terminology in SWEBOK

The concept of *quality,* together with its set of multiple subconcepts, is used in all Knowledge Areas of SWEBOK, as illustrated by the number of entries of the term 'quality' in each of the SWEBOK chapters (Table 1). In this section, we analyze how the term 'quality' and its related subconcepts are used in the context of the SWEBOK Guide – Trial Version 1.00.

Table 1: 'Quality' in the 10 SWEBOK Knowledge Areas

Knowledge Area	**The Number of times 'quality' is mentioned**
Software Requirements	60
Software Design	21
Software Construction	9
Software Testing	16
Software Maintenance	22
Software Configuration Management	19
Software Engineering Management	32
Software Engineering Process	16
Software Engineering Tools and Methods	4
Software Quality	187
Appendix and Introduction	58

Table 2 presents a more detailed inventory of the statements in which the term 'quality' appears in the Testing chapter. On the left-hand side, the section of the chapter is mentioned, and in the middle column we have indicated whether the term 'quality' has been used alone, or as part of a related expression containing the 'quality' term. This Table 2 illustrates how the two authors of this chapter have used '*quality'* and its subconcepts, both in a generic meaning of quality and also in the particular context of software engineering, that is, 'software quality'. Another, more extensive illustration of the use of quality-related statements in the Requirements chapter is presented in the Appendix.

Quality and its subconcepts in the generic sense are defined in multiple sources [2]. Often when *quality* is used in SWEBOK, the particular meaning is *software quality*. In this paper, whenever we find a term (or an expression) which has a different contextual meaning (even though the same term is used), we will refer to this as a distinct 'concept'. For the sake of clarity and consistency, it will therefore be necessary to find out which concepts are used in SWEBOK and how they have been defined in particular contexts.

Table 2: Inventory of quality concepts in the SWEBOK chapter on Testing

Software Testing	Expression	SWEBOK Statements
Introduction (p. 5-1)	product quality	Testing is an important, mandatory part of software development; it is a technique for evaluating product quality and also for indirectly improving it, by identifying defects and problems.
Introduction (p. 5-1)	quality	As more extensively discussed in the Software Quality chapter of the Guide to the SWEBOK, the right attitude towards quality is one of prevention; it is obviously much better to avoid problems than to repair them.
Introduction (p. 5-1)	quality product	It is perhaps obvious, but worth stating, that, even after successfully completing an extensive testing campaign, the software could still contain faults; also, defect-free code is not a synonym for product quality.
Introduction (p. 5-1)	quality analysis	In the Software Quality (SQ) chapter of the Guide to the SWEBOK already referred to, activities and techniques for quality analysis are categorized into: static techniques (no code execution) and dynamic techniques (code execution).
Introduction (p. 5-1)	product quality	Although this chapter focuses on testing, that is, dynamic techniques, static techniques are equally important for the purposes of evaluating product quality and finding defects.

Breakdown of Topics for the Software Testing Knowledge Area (p. 5-7)	quality analysis	However, a comprehensive view of the Knowledge Area of Software Testing as a means for evaluating quality must include other, equally important testing objectives, e.g. reliability measurement, usability evaluation, contractor's acceptance, for which different approaches would be taken.
Breakdown of Topics for the Software Testing Knowledge Area (p. 5-11)	software quality	It is also informative to consider testing from the point of view of software quality analysts, users of CMM and Cleanroom processes, and of certifiers.
Breakdown of Topics for the Software Testing Knowledge Area (p. 5-11)	quality analysis	Measurement is instrumental in quality analysis.
Breakdown of Topics for the Software Testing Knowledge Area (p. 5-11)	quality measurement	Wider coverage of the topic of quality measurement, including fundamentals, measures and techniques for measurement, is provided in the Software Quality chapter of the Guide to the SWEBOK.
Breakdown of Topics for the Software Testing Knowledge Area (p. 5-11)	quality prediction	This information can be very useful in making quality predictions as well as for process improvement.
Breakdown of Topics for the Software Testing Knowledge Area (p. 5-12)	software quality	These (what?) should foster a common culture towards software quality, by which early failure detection becomes an objective for all those involved, not only testers.
Breakdown of Topics for the Software Testing Knowledge Area (p. 5-13)	quality assurance	Execution of tests is generally performed by testing engineers with oversight by quality assurance personnel and, in some cases, customer representatives.

The term 'quality' is therefore quite extensively referenced in the SWEBOK Guide, from numerous viewpoints, to represent various facets (concepts and related subconcepts) of quality. However, are these distinct facets of quality labeled distinctly and unambiguously, or are their interpretations highly textually dependent? How can we ensure in particular that beginner users of this Guide properly recognize such distinct facets when they are not properly labeled?

This issue of consistency of terminology is particularly important in the development of international standards, and specific techniques have been developed to improve the consistency of the terminology within each standard. For instance, it is mandatory in a standard to provide, in a predetermined section, all definitions, carefully crafted, and to allow no redefinition within the body of the text. During the review cycles of the draft versions of these standards, experienced reviewers have developed a few verification criteria, such as: for each term defined in the official definition section, there must be no

further 'is defined' within the main body of the standard, and: no such term can be further redefined within the text through related expressions like 'is…', 'is used to…', 'uses…', 'is defined as…'. To improve consistency, redefinitions within the text must be withdrawn if they correspond exactly to the official definition, or be relabeled as distinct expressions if they convey a distinct concept or subconcept not specifically stated in the official definition.

This practitioner's approach to standards development and review is modeled in Figure 2, and was used in the initial iteration of our inventory of the term *quality* in the SWEBOK Guide. However, this approach was not used initially to improve the consistency of the SWEBOK Guide, but for another purpose, which was to identify, and recognize, the full set of concepts and sub-concepts about *quality* in the SWEBOK Guide. Indeed, since the SWEBOK had already been extensively been reviewed chapter by chapter by world experts, we were not specifically searching for duplication or inconsistencies, but rather to identify, and inventory, all the distinct viewpoints of *quality* being presented and discussed.

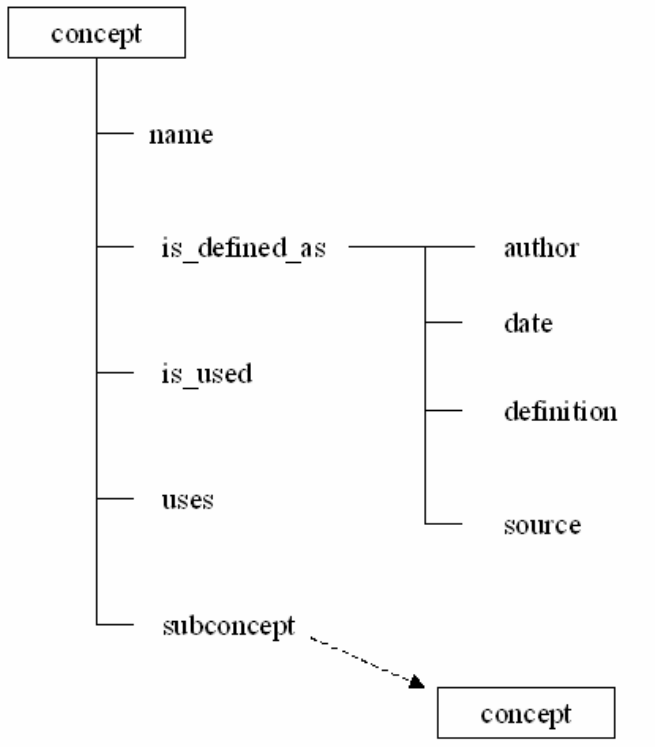

Figure 2: Practitioners' approach to recognizing distinct uses of subconcepts about a single concept

Figure 3 and 4 presents both the results of our inventory, and partial structuring, of the quality concepts identified within the whole SWEBOK Guide using the approach illustrated in Figure 2. Figure 3 lists all concepts and sub-concepts identified for the term 'quality' in the generic sense, and Figure 4 for the term in the particular context of 'software quality'.

The view in Figure 3 shows that *software quality* (bottom center) is one particular subconcept of *quality*. Four other subconcepts of quality have their own subconcepts; for example, *process quality* is associated with the subconcept *process quality assessment* (similarly for: document quality, quality analysis and product quality). Three subconcepts have no referenced parents in the Guide (bottom right-hand side of Figure 4): *quality evaluation tools, quality improvement paradigm and construction quality assurance*.

Figure 4 shows the subconcepts of *software quality*. *Software quality* has nine subconcepts and this is many fewer than *quality* by itself in the generic sense.

Of course, the structuring of concepts and sub-concepts in Figure 3 and 4 is only preliminary. Further work will be required later on to improve and optimize this initial structuring; it should, however, be useful in the current review cycle of the SWEBOK Guide. In Figure 4 we illustrate also an initial structuring of multiple inheritances in this area: for example, "software quality assurance" would be subconcepts of both "quality assurance" and "software quality".

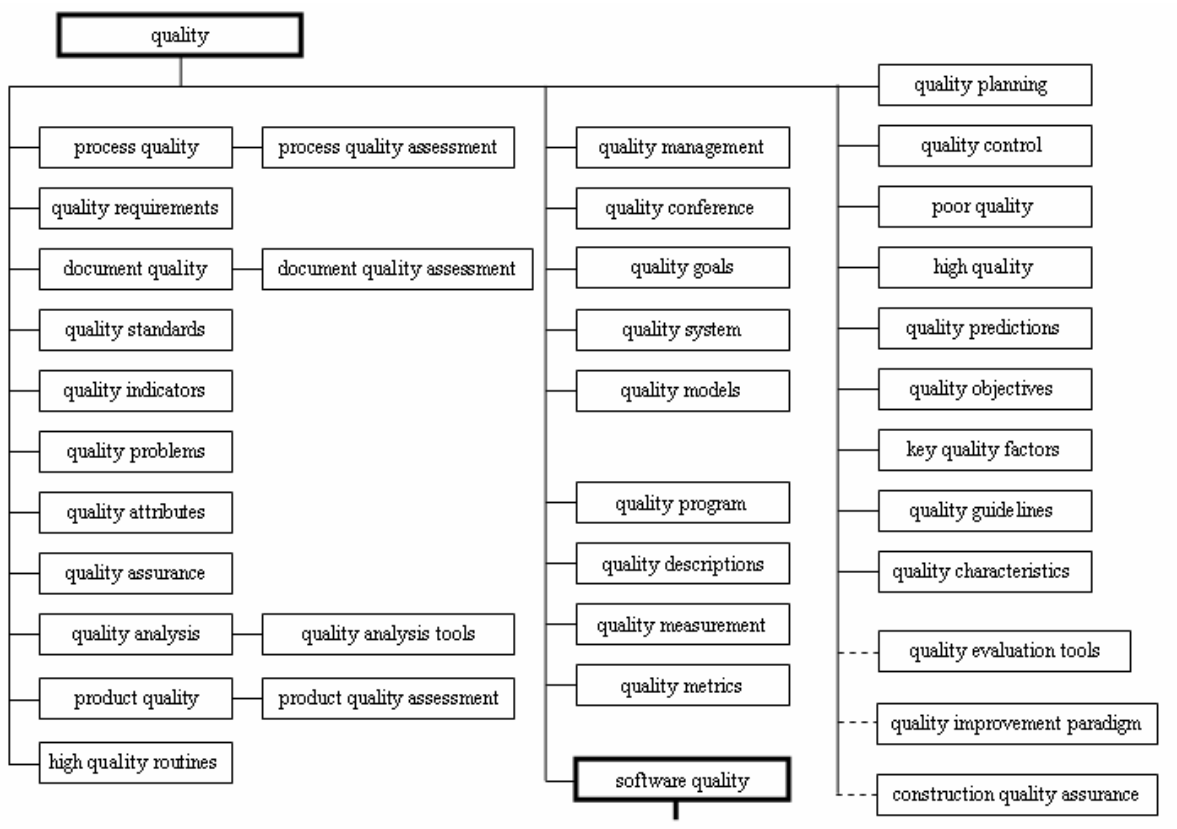

Figure 3: Initial structuring of subconcepts of quality used in SWEBOK

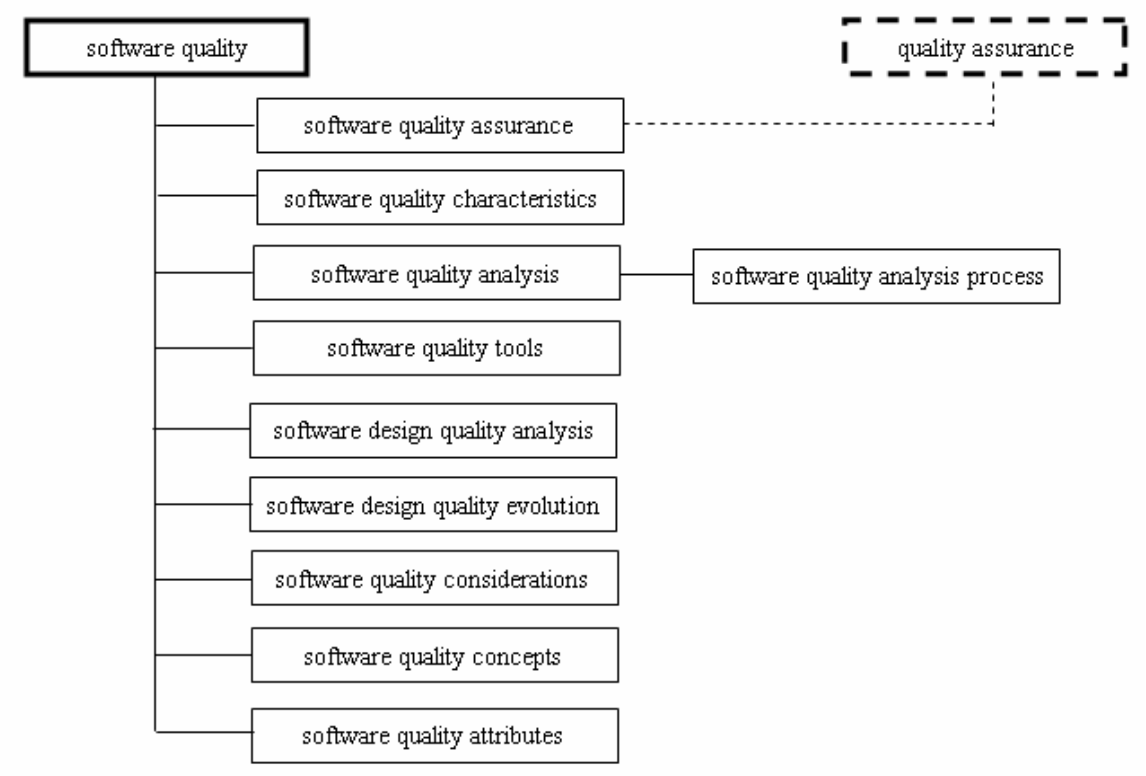

Figure 4: Subconcepts of 'software quality' used in SWEBOK

In the inventory and analysis of the SWEBOK Guide, it was observed that sometimes expressions used in a particular sense were being replaced by generic expressions, leaving the reader to figure out from the context that the expression was being used in the particular sense. One example is the use of the subconcepts *quality attributes* and *software quality attributes*; both appear in SWEBOK. 'Quality attributes' is used 15 times and 'software quality attributes' twice. Table 3 shows that their meanings are often the same.

Table 3: Context of use of 'quality attributes' and 'software quality attributes' in SWEBOK

Number id.	Chapter	Statements in the SWEBOK Guide text
1	Introduction	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.
2	Software Requirements	Of particular interest are issues of ***software quality attributes*** and measurement, and software process definition.
3	Software Engineering Management	Quality management – quality is defined in terms of pertinent attributes of the specific process/project and any associated product(s), perhaps in both quantitative and qualitative terms. (These ***quality attributes*** will have been determined in the specification of detailed requirements.)

4	Software Quality	The software engineer, in discussing ***software quality attributes*** and the processes necessary to ensure their presence, should keep in mind the value of each attribute and the sensitivity of the value of the product to changes in it.
5	Software Quality	***Quality attributes*** may be present or absent, or may be present to a greater or lesser degree, with tradeoffs among them, and with practicality and cost as major considerations.
6	Software Quality	Terminology for ***quality attributes*** differs from one taxonomy or model of software quality to another; models may have different numbers of hierarchical levels and different total numbers of attributes.

3 SWEBOK terminology and ontology

Detailed analysis of the SWEBOK text reveals (as previously illustrated with the term 'quality') that terms and expressions (concepts and related subconcepts) are often used in chapters with both similar and dissimilar meanings. Of course, this is only one example; we also analyzed the terms 'measurement', 'defect', 'validation' and 'verification' with the same results.

For users (humans or machines), different interpretations in distinct contexts sometimes make the meanings of terms confusing and ambiguous, while a coherent terminology adds clarity and facilitates understanding. “People can’t share knowledge if they don’t speak a common language” [4]. “[The need to define] domain-specific vocabulary is a major barrier to knowledge base construction” [17].

The development, and evolution, of an international consensus on a topical access to the existing knowledge of software engineering is one of the five objectives of the SWEBOK project, and the basic need for the construction of SWEBOK. Readers are reminded that such knowledge about software engineering has been developed by a large number of researchers and practitioners, from multiple viewpoints, initially without commonalities in terminology across topics and subtopics within this domain of knowledge.

A terminology, as a general term for all kinds of controlled vocabularies, can help to clear up ambiguities in the terms used in the context of software engineering. Of course, the IEEE has its own Standard Glossary of Software Engineering Terminology, and other terminologies also exist in the area of Computer Science [8] [9]. Ontologies define a common vocabulary for researchers who need to share information in a domain [15]. For researchers, an ontology will also include machine-interpretable definitions of basic concepts in the domain and the relations among them. The ontology approach seems a promising path to follow to tackle terminology issues at lower levels of detail, since an ontology provides a standard terminology for a specific context.

In recent years, the development of ontologies has been moving from the realm of Artificial-Intelligence laboratories to the desktops of domain experts. Many disciplines now develop standardized ontologies that domain experts can use to share and annotate information in their fields. Medicine for example, has produced large, standardized and structured vocabularies, and there is also evidence of emerging ontologies in the field of software engineering [3]. For Gruber, “an ontology is a specification of a conceptualization” [6].

An ontology is also a specification of some topic [17]. It is a formal and declarative representation which includes the vocabulary required for referring to the concepts in that subject area and the logical statements that describe what the concepts are, how they are related, and can be related, to one another. Ontologies therefore provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships which hold among the concepts in that vocabulary.

Some of the reported benefits of ontologies are that they [15]:
Enable a new and effective way to reuse knowledge;
Help us use, and understand, some area of knowledge better;
Separate fundamental knowledge from operational knowledge;
Help us analyze the structure of knowledge;
Help us reach a consensus on our understanding of some area of knowledge;
Help us share a common understanding of the structure of information, among people or software agents;
Enable a machine to use the knowledge in some application.

In addition, a SWEBOK ontology could help to separate software engineering knowledge from other operational knowledge. In this way, general statements could be consciously delimited. For example, every product has quality attributes, however the ontology shows that quality attributes in the context of software (software quality attributes) are different from quality attributes for other products.

An internationally recognized software engineering ontology, when and if one becomes available, would make it easier to carry out changes in the knowledge and to teach this new knowledge to software engineers. In addition, explicit specifications of software engineering knowledge are useful for new researchers who will learn the meaning of concepts in the domain.

Ontology development is necessarily an iterative process. Concepts in the software engineering ontology should be close to objects of interest (physical or logical) and to the relationships between them.

4 Design of a SWEBOK ontology

The first challenge in developing an ontology for SWEBOK is to define what the ontology should contain and what it should be used for.

Of course, the SWEBOK Ontology should include all the important concepts about software engineering. These concepts should be supported by widely accepted definitions, facilitating common understanding by all users in this domain of knowledge. Concurrently, an ontology should provide a necessary delimitation with respect to other domains of knowledge. In practice, developing an ontology also includes defining classes within the ontology and arranging the classes in a taxonomic (subclass–super class) hierarchy. The structure of knowledge provided in the SWEBOK Guide provides a starting point for the design of a software engineering ontology.
SWEBOK is the super class of the ontology. The ten Knowledge Areas are the subclasses of the super class and represent specialized views of parts of the software engineering knowledge. Each Knowledge Area is represented by a structured set of concepts and corresponding definitions. All concepts are subclasses of the super class and they can also be subclasses of one or more Knowledge Areas.
A second important aspect in the design of an ontology is that much of the knowledge of software engineering in the SWEBOK Guide is represented by links to internal and external references. To model such links, we need more than only the unidirectional HTML links. In the current book format of the SWEBOK Guide, it is not possible for the user to access a single (and unique) reference for a concept. In future versions of the Guide, the user should ideally be provided with a quick way to find either a reference or a concept by means of the SWEBOK ontology, as well as additional information about his search.

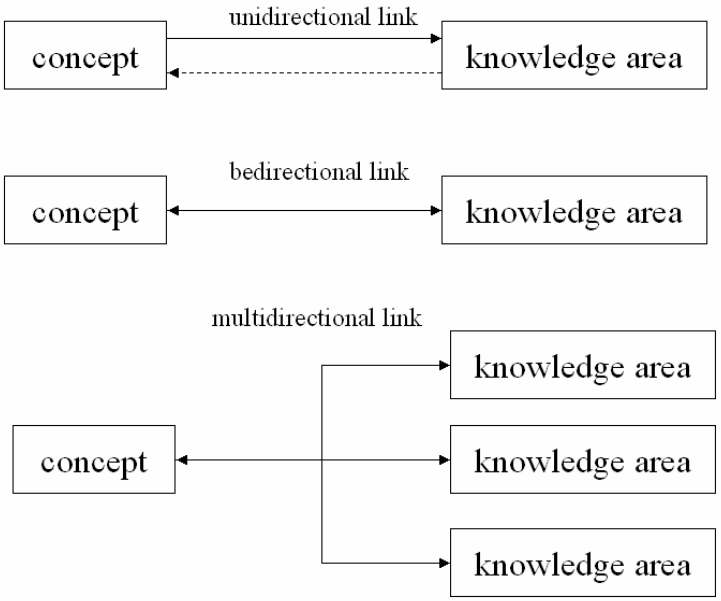

Figure 5: Different types of links between concepts and SWEBOK Knowledge Areas

Bidirectional and multidirectional links allow information-sharing in both directions, which means that every concept can by referenced in one or more ways. Also, the path from the reference to the concept is available, as illustrated in Figure 6 for the testing concept and some of the relevant references.

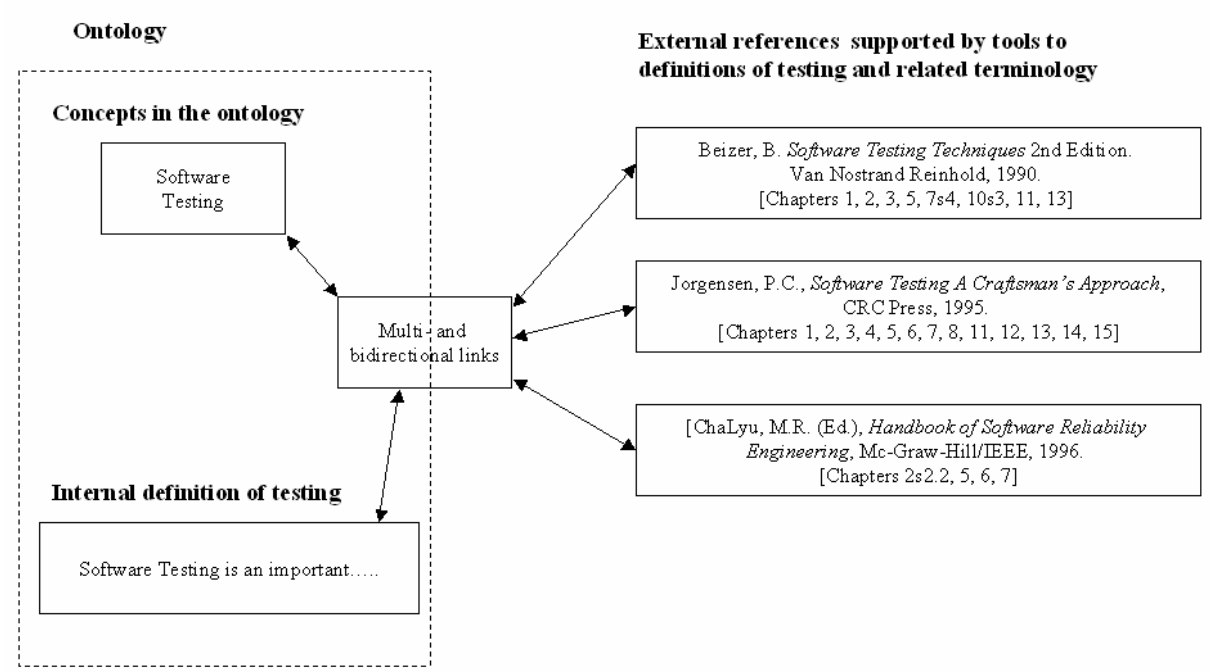

Figure 7: Example of a link structure with internal and tool-supported external sources

An ontology with bidirectional and multidirectional links would make it possible for every user (as well as applications and agents) to have very fast access to the corresponding details of high-level knowledge.
By contrast to other ontologies, such as in the medical field, the structure of a software engineering ontology will be relatively flat. Under the root element, there will be different Knowledge Areas and different concepts. In other domains, an electronic marketplace, for example, the structure would be much deeper.

The SWEBOK structure will contain many different links to help the user find knowledge quickly, and most of these links will point to external references (see Figure 8).

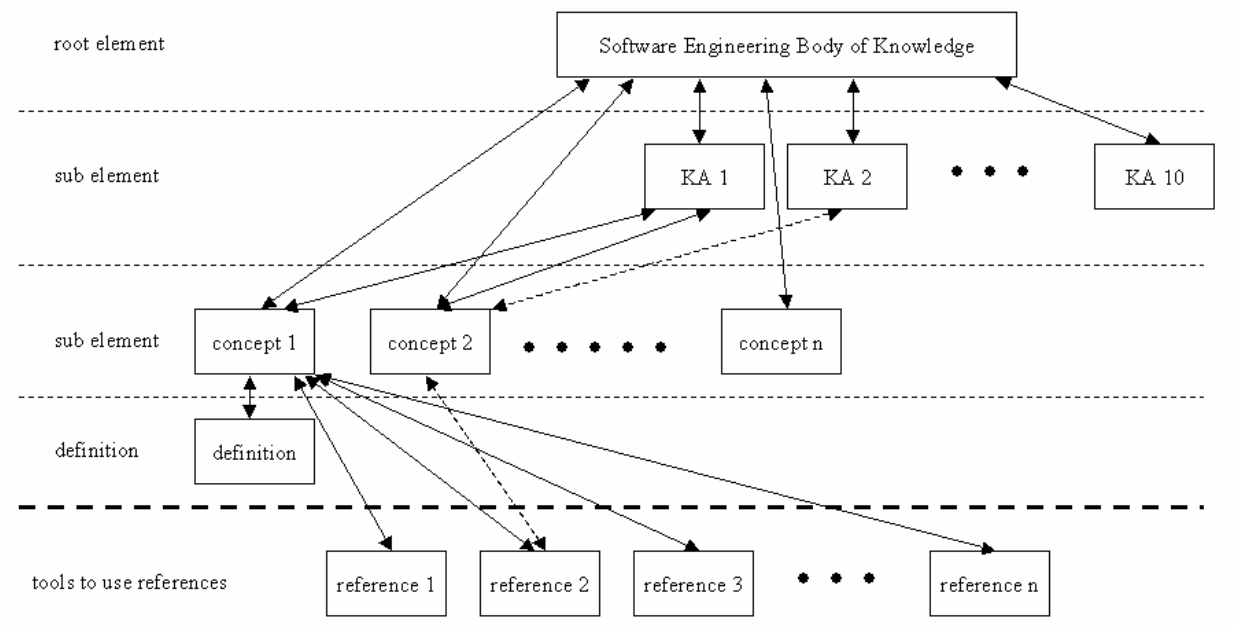

Figure 9: Design of a software engineering ontology with different levels of knowledge

Only a few links are illustrated in Figure 9, out of a much larger number of available references. All concepts are subclasses of SWEBOK and also subclasses of one or more Knowledge Areas. Every concept has a definition and one or more internal or external references. To find the knowledge he is looking for, the user can use various views. For example, he can navigate from Knowledge Areas through to concepts with their definitions and references. He also can search from the perspective of all the concepts in SWEBOK. In the future, if bidirectional links are available, users will also be able to navigate from a reference to a Knowledge Area or to other references.

An initial example of a data structure for a software engineering ontology is presented in Figure 10. Under the root element (SWEBOK) are the Knowledge Areas with their names and a list of all the concepts used in each of the corresponding Knowledge Areas. Also under the root element are all the concepts used in all the Knowledge Areas. These concepts have the following attributes: 'name', 'is_defined_as', 'is_used' and 'uses'. The definition of the concept gives information about the source of the definition. The expression 'is_used' represents a list of all the Knowledge Areas that use the concept, and 'uses' represents a list of references outside SWEBOK which are supported by tools.

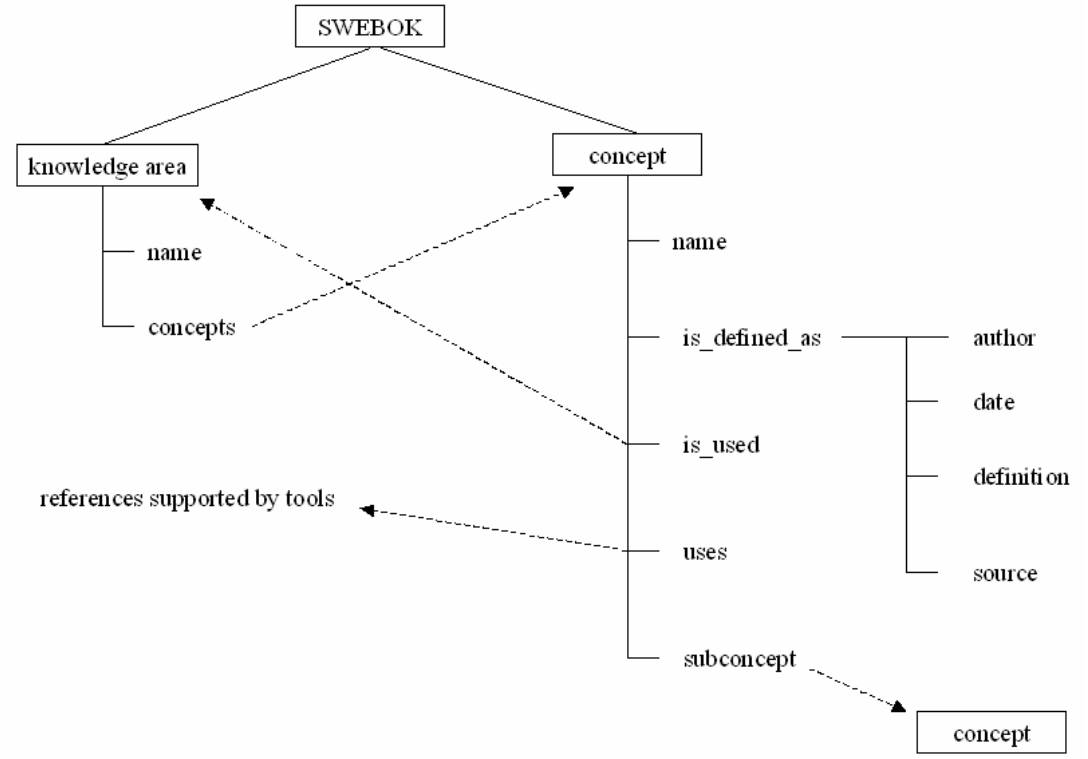

Figure 10: Structure for a software engineering ontology

Concepts can also have associated subconcepts. These subconcepts are also concepts, and have the same structure. For example: software quality is a subconcept of quality, and design quality is also a subconcept of quality.

Various technologies are available for developing a SWEBOK ontology [3] [16] [18]. All are based on the Extensible Markup Language (XML) a simple, very flexible text format derived from the Standard Generalized Markup Language (SGML) [10].

5 Conclusions and Future Work

Under the leadership of the IEEE-Computer Society, a compendium and guide to the software engineering body of knowledge has been developed and should be approved as ISO/IEC TR 19759 by the end of 2003. A very detailed inventory of terms and expressions used in the SWEBOK Guide indicates, however, a need to improve the consistency of the terminology. The current Trial version of the guide represents a large number of viewpoints in a domain where there is not yet a consensus on a single set of software engineering terms. Through an example with the use of the term 'quality' (or *quality*) in the 2001 version of the SWEBOK Guide, we have illustrated the need to improve the coherency of the terminology.

The design of a software engineering ontology could help improve the consistency of the terminology in the SWEBOK Guide. An ontology is a flexible and useful way to define terms, their concepts and subconcepts, and show how they are related to one another in the context of domain knowledge.

An ontology is also a conceptualization, and presents domain knowledge and its structure in a general manner.

In this paper, we presented a candidate approach for the design of an ontology for SWEBOK. Various languages are available to create an ontology, and all are based on the Extensible Markup Language (XML). To decide which is the best one to use is not a simple question, because ontologies and their languages are just in the beginning stages of development, as are their technologies. A critical step in creating a SWEBOK ontology will be to establish which technology will best support it.

To help the developers of the ontology, research tools are currently being developed to help create the data structure. It will be very useful, for instance, to identify fundamental terms and concepts and to have available text extraction and rule extraction tools to facilitate the process. Research is progressing well on knowledge extraction tools (inference engine) to help in the design of an ontology based on the SWEBOK knowledge and in finding new knowledge.

References

[1] A. Abran, J. Moore, P. Bourque, R.L. Dupuis, L. Tripp, *Guide to the Software Engineering Body of Knowledge – SWEBOK*, Trial Version 1.0, IEEE-Computer Society Press, May 2003, URL: http://www.swebok.org

[2] ANSI/IEEE STD 1061, *IEEE Standard for a Software Quality Metrics Methodology*, IEEE Computer Society Press, New York 1998.

[3] *The DARPA Agent Markup Language Homepage*, July 2003, URL: http://www.daml.org/

[4] T. H. Davenport, L. Prusak, *Working Knowledge: How organizations manage what they know*, Harvard Business School Press, 1997.

[5] N. Guarino, P. Giaretta, *Ontologies and Knowledge Bases: Towards a Terminological Clarification*, In N.J.I. Mars, editor, *Proc*. of the 2nd Intern. Conf on Building and Sharing Very Large Knowledge Bases. IOS Press, Enschede, The Netherlands, 1995.

[6] T. R. Gruber. *A Translation Approach to Portable Ontology Specifications*, Stanford University, April 1993.

[7] J. Hasebrook, L. Erasmus, G. Doeben-Henisch, *Knowledge Robots for Knowledge Workers: Self-Learning Agents Connecting Information and Skills*, In: Jain/Chen/Ichalkaranje: Intelligent Agents and Their Applications, Physica-Verlag Heidelberg New York, Heidelberg, 2002, pp.59-81.

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

[9] IEEE Standard Glossary of Application Terminology, IEEE, Piscataway, NJ, IEEE Standard 610.2-1987, 1987.

[10] ISO 8879:1986, *Information Processing - Text and Office Systems - Standardized Generalized Markup Language (SGML),* International Organization for Standardization, Geneva, 1986.
[11] ISO, ISO/IEC 9126-1:2001 *Software engineering – Product quality – Part 1:Quality model*, International Organization for Standardization/International Electrotechnical Commission., Geneva, 2001.
[12] ISO, ISO/IEC 15939:2002 *Software Engineering: Software Measurement Process*, International Organization for Standardization/International Electrotechnical Commission., Geneva, 2002.
[13] ISO, International Vocabulary of Basic and General Terms in Metrology, International Organization for Standardization – ISO, Geneva, 1993.
[14] *The KAON open-source ontology applications Homepage*, May 2003, URL: http://kaon.semanticweb.org/
[15] N. F. Noy and D. L. McGuinness, *A Guide to Creating Your First Ontology*, Stanford University, Mai 2003, URL: http://protege.stanford.edu/publications/ontology_development/ontology101.pdf
[16] *OIL Homepage*, July 2003, URL: http://www.ontoknowledge.org/oil/
[17] Stanford KSL Network Services, June 2003, URL: http://www-ksl-svc.stanford.edu:5915/doc/network-services.html
[18] *W3C Homepage*, July 2003, URL: http://www.w3.org/
[19] N. Zhong, *Ontologies in Web intelligence*, In: Jain/Chen/Ichalkaranje: Intelligent Agents and Their Applications, Physica-Verlag Heidelberg New York, Heidelberg, 2002, pp.83-97.

Annex

Table 4: Inventory of quality concepts in 2 SWEBOK chapters: Software Requirements and Software

SWEBOK chapter and sections	used concept	Quality-related statement
Software Requirements		
Definition of the Software Requirements Knowledge Area (p.2-2)	quality requirements	Non-functional requirements are sometimes known as constraints or quality requirements.
Definition of the Software Requirements Knowledge Area (p.2-5)	quality	Instead, requirements typically iterate toward a level of quality and detail that is sufficient to permit design and procurement decisions to be made.
Definition of the Software Requirements Knowledge Area (p.2-5)	requirements quality	However, requirements engineers are necessarily constrained by project manage-ment plans and must therefore take steps to ensure that the requirements quality is as high as possible given the available resources.
Breakdown of Topics for Software Requirements (p. 2-9)	process quality assessment	This subtopic is concerned with requirements engineering process quality assessment.
Breakdown of Topics for Software Requirements (p. 2-9)	quality standards	It will help to orient the requirements engineering process with quality standards and process improvement models for software and systems.
Breakdown of Topics for Software Requirements (p. 2-9)	process quality	Process quality and improvement is closely related to the software quality KA and the software process KA.
Breakdown of Topics for Software Requirements (p. 2-9)	process quality	The process quality and improvement subtopic is concerned with quality.
Breakdown of Topics for Software Requirements (p. 2-11)	product quality	The quality of requirements elicitation has a direct effect on product quality.
Breakdown of Topics for Software Requirements (p. 2-13)	quality, product quality, software quality	The quality of the analysis directly affects product quality. In principle, the more rigorous the analysis, the more confidence can be attached to the software quality.
Breakdown of Topics for Software Requirements (p. 2-13)	quality	This topic is concerned with the structure, quality and verifiability of the requirements document.
Breakdown of Topics for Software Requirements (p. 2-13)	quality, product quality	The quality of the requirements document can dramatically affect the quality of the product.
Breakdown of Topics for Software Requirements (p. 2-13)	quality indicators	A number of quality indicators have been developed that can be used to relate the quality of an SRS to other project variables such as cost, acceptance, performance, schedule, reproducibility, etc.

Breakdown of Topics for Software Requirements (p. 2-14)	quality	The quality of the requirements documents dramatically affects the quality of the product.
Breakdown of Topics for Software Requirements (p. 2-14)	quality attributes	Quality attributes of requirements documents can be identified and measured.
Breakdown of Topics for Software Requirements (p. 2-14)	quality problems	These include the danger of users' attention being distracted from the core underlying functionality by cosmetic issues or quality problems with the prototype.
Breakdown of Topics for Software Requirements (p. 2-14)	quality	The quality of the models developed during analysis should be validated.
Breakdown of Topics for Software Requirements (p. 2-14)	quality	Validation is all about quality - the quality of the requirements.
Breakdown of Topics for Software Requirements (p. 2-15)	quality system	A naming scheme for generating these IDs is an essential feature of a quality system for a requirements engineering process.
Breakdown of Topics for Software Requirements (p. 2-15)	software quality	The availability of modern requirements management tools has improved this situation and the importance of tracing (and requirements management in general) is starting to make an impact on software quality.
Breakdown of Topics for Software Requirements (p. 2-15)	quality	Requirements management is a level 2 key practice area in the software CMM and this has boosted recognition of its importance for quality.
Breakdown of Topics for Software Requirements (p. 2-15)	process quality assessment	We believe this topic adds great value to any discussion of requirements engineering as it is directly concerned with process quality assessment.
Breakdown Rationale (p. 2-16)	document quality assessment	The breakdown is similar to that discussed in most texts, apart from document quality assessment.
Breakdown Rationale (p. 2-16)	product quality assurance	The relationship of requirements engineering product quality assurance, tools and standards is provided in the breakdown.