UQÀM
Université du Québec à Montréal
Doctorat en Informatique cognitive

Université du Québec
École de technologie supérieure

Issues in the development of an ontology for an emerging engineering discipline

Authors

Mendes, O. (UFPB-DECOM, UQAM-DIC)
Abran, A. (ETS-DGL&TI)
Pezzin, J. (UFES)

17th SEKE - International Conference on Software Engineering and Knowledge Engineering

Taipei, July 14 – 16, 2005

Presentation Outline

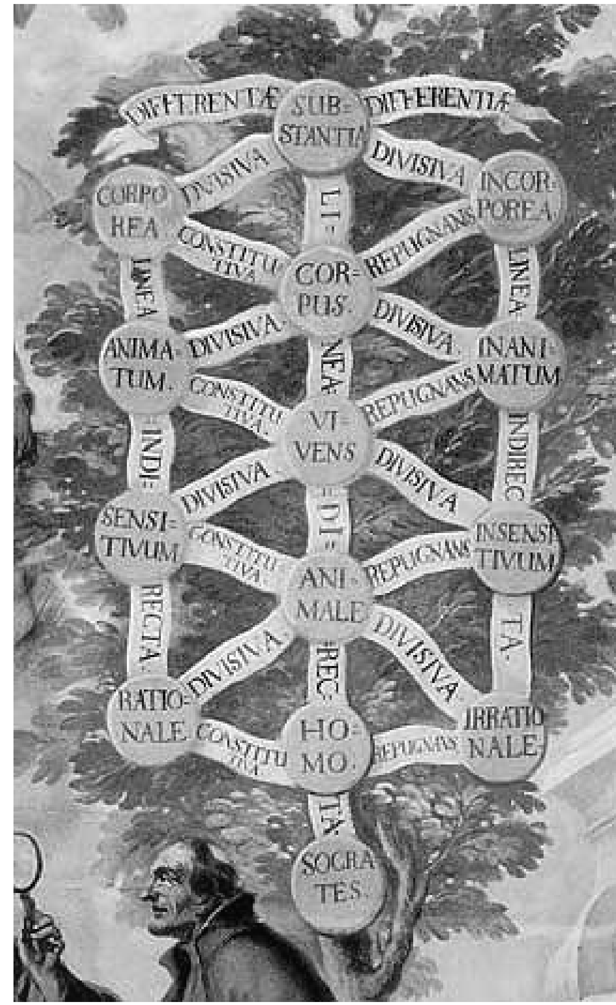
- Introduction
- Problem
- Justification
- SWEBOK Project
- Software Engineering Ontology
- Ontology Development Process
- V&E (Validation and Extension) Strategy
- Results
- Software Engineering Ontology: An Application
- Summary and Research Contributions
- Future Work

Introduction (1/4)

Ontology in philosophy:

- A discipline of philosophy since Aristotle 384-322 BC (Catégories) and Porphyry 233-310 AD (Isagoge)
- A study of being *qua* being
(view in a very general perspective)
- A study of the nature of possible:
What entities possibly exist ?
- Study of the nature of possible:
What are the nature and most general characteristics of the entities that we recognize as existent?

(Guarino et Welty, 2000)



Porphyry (233-310 AD)

Introduction (2 / 4)

Ontology in computer domain:

- Introduced in the early 90s with the DARPA project «Knowledge Sharing Initiative» (Patil et al.,1992);
- Goal: to reduce the time, effort and costs required to develop knowledge data bases, through sharing and reuse (Neches et al., 1991);
- Since we cannot share and reuse knowledge if we do not speak the same language and have somehow a consensus concerning the meanings of the concepts used to communicate, ontologies were introduced to describe the semantics and to make explicit the domain assumptions associated to the knowledge to be shared or reused (Davenport, Thomas H., 1993; Guarino et Schneider, 2002).

Introduction (3 / 4)

- **Ontology in the computer domain:**

  A formal explicit description of a consensual shared understanding of the pertinent entities (and their interrelationships) considered as existing in a certain domain of knowledge, and the terms we use to refer to them and their agreed meanings and properties (Gruber,1993; Rector et al., 2004).

- Ontologies make thus possible communication among:

  - People/organizations,
  - Systems/software agents
  - People and systems

  by agreeing and sharing a common understanding about a conceptualisation, recognizing the existence of a set of objects and their relationships, as well as the terms used to refer to them and their agreed meanings (ontological commitment).

  (Guarino et Schneider, 2002; Rector et al., 2004).

Introduction (4/4)

Ontology in the Software Engineering:

- Provide a source of precisely defined terms that can be communicated across people, organisations and applications (information systems or intelligent agents);
- Offer a consensual shared understanding concerning the domain of discourse;
- Render explicit all hidden assumptions concerning the entities pertaining to a certain domain of knowledge.

(Gruber,. 1993; Gruninger et Lee, 2002; Garzás J., Piattini M. 2005)

Problem

- **Despite some initial effort to develop partial ontologies**
  - Software maintenance (Kitchenham, B., et al. 1999; Ruiz et al., 2004);
  - Software measurement (Martin et Olsina, 2003);
  - Software quality (Wille et al., 2003; 2004);
  - OO Design (Garzás J., Piattini M. 2005);

Software Engineering as a field of knowledge, still does not have a comprehensive detailed ontology which describes the concepts that domain experts agree upon, as well as their terms, definitions and meanings.

Justification

- The development of a "**software engineering domain ontology**" would allow :
  - Provide a formal representation of the body of knowledge of the Software Engineering discipline;
  - Share and reuse knowledge accumulated until now in the Software Engineering field;
  - Open new avenues to automatic *validation* and *interpretation* of this knowledge using information systems or *intelligent* software agents.

SWEBOK Project (1/2)

- **Participants:**
  - IEEE Computer Society – UQAM – ETS – ISO
  - Over 500 reviewers from the industrial and academic fields, government agencies, professional societies, international standards organisations, and research centers.

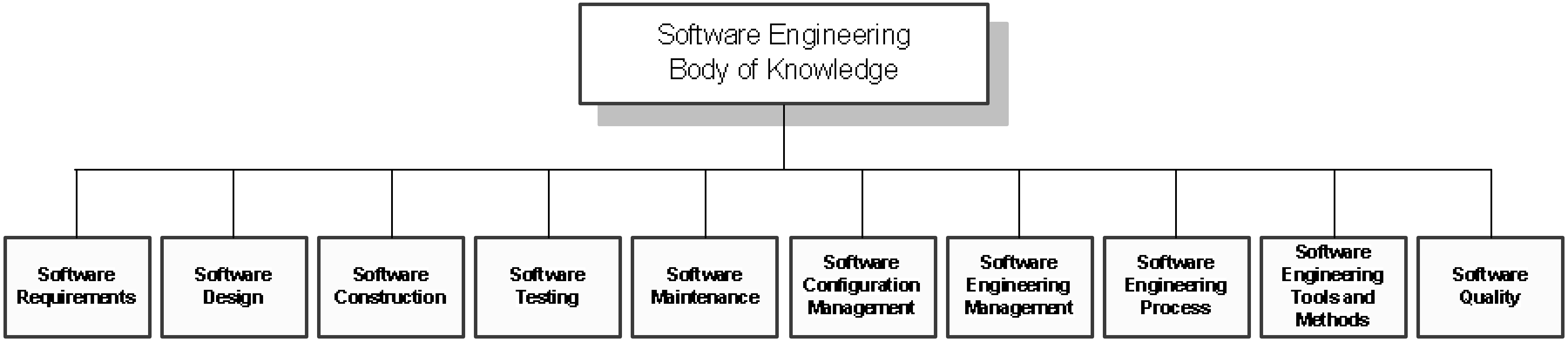
- **Goals :**
  - To characterize the content of the software engineering discipline;
  - To provide topical access to the software engineering body of knowledge;
  - To promote a consistent view of software engineering worldwide;
  - To clarify the place – and set the boundaries – of software engineering with respect to other disciplines (such as computer science, project management, computer engineering, and mathematics);
  - To provide a foundation for curriculum development and individual certification material.

(Abran 2000; Abran et al., 2000, 2000a, 2000b)

SWEBOK Project (2 / 2)

- The SWEBOK Project (Software Engineering Body of Knowledge) developed progressively a consensus concerning:
  - **The knowledge domains contained within Software Engineering;**
  - **Their contents and the main references constituting the body of knowledge**;
  - **The scientific disciplines participating in each knowledge area.**

- The resulting product of the SWEBOK project it is not the body of knowledge itself, but rather a guide to it, permitting to gain consensus on the core subset of knowledge characterizing the software engineering discipline (Bourque, Dupuis, Abran, 1999; Abran, Moore et al., 2005).



The Software Eng. Ontology

- The SWEBOK project has allowed a progressive *building of consensus* among the experts participating to the Delphi panels, concerning the knowledge domains contained within of the Software Engineering discipline and their content;

- The SWEBOK Guide represents an important and privileged information source for the construction of a Software Engineering domain ontology, containing *validated* and *consensual* domain knowledge;

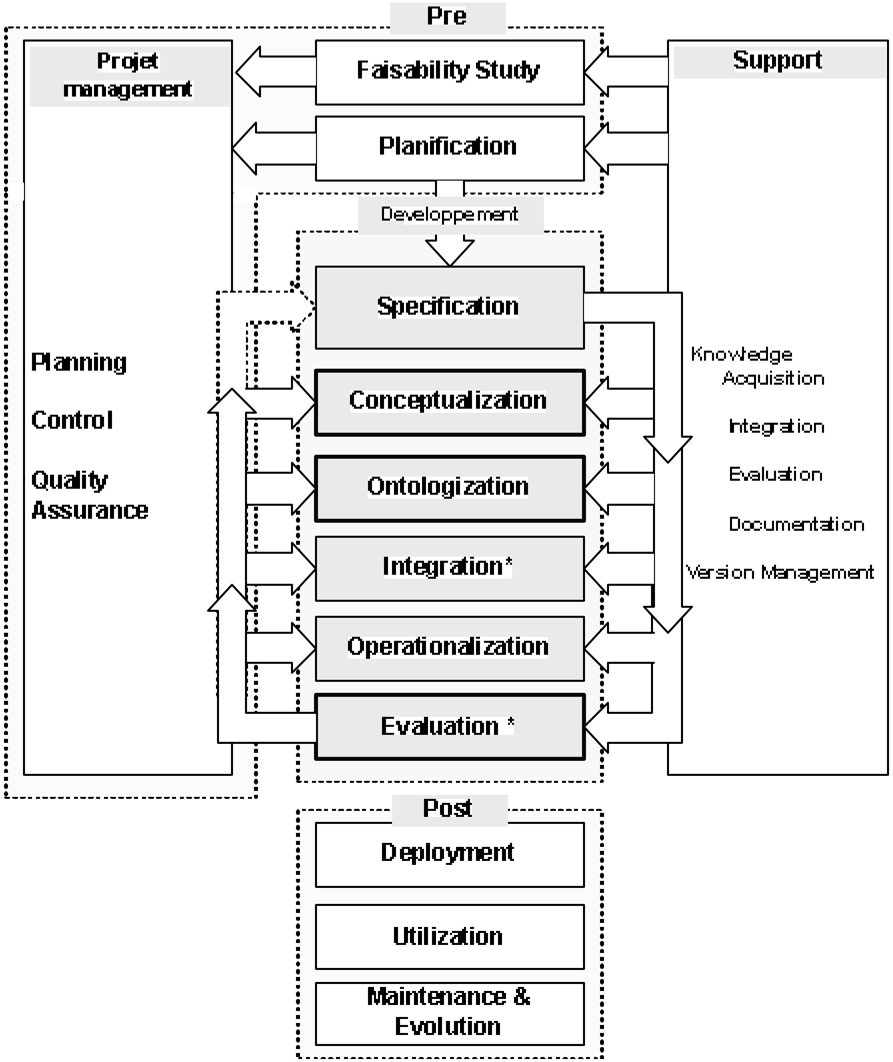
- Our approach to build a domain ontology for the Software Engineering using as primary information sources:
  - **The SWEBOK Guide** (Feb, 2005 version)
  - **Technical standards** (ex: 610.12-1190 IEEE; ISO/IEC 12207-95).

The Ontology Development Process

(1 / 5)

- Specification
- Conceptualization
- Ontologization
- Integration
- Operationalization
- Evaluation

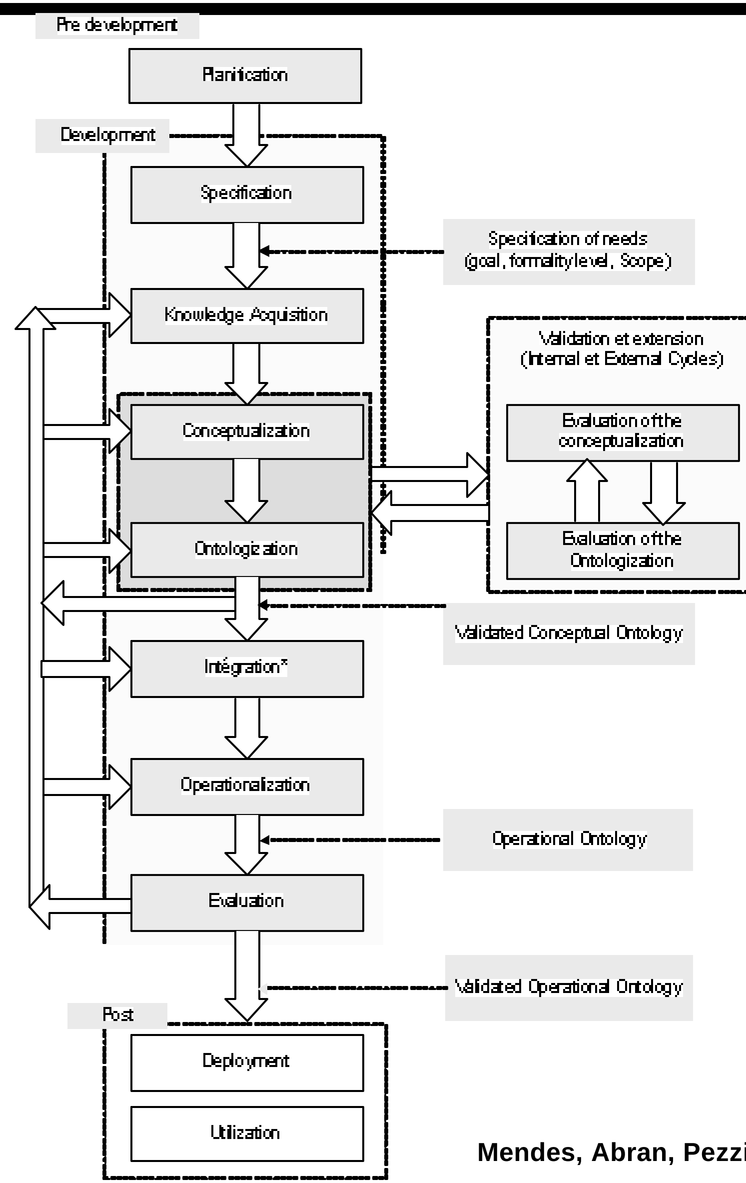
(Mendes, 2004)



The Ontology Development Process

(2 / 5)

- Our approach to develop the Software Engineering domain ontology requires three main phases:
  - **Proto-ontology construction** *Protos* (Greek): seed, first;
  - **Proto-ontology validation and extension (V&E) cycles**
  - **Ontology Operationalization and Evaluation.**



The Ontology Development Process

(3 / 5)

Proto-ontology construction:

- We started the ontology construction process with the development of a proto-ontology using the information contained in the SWEBOK Guide;

- This proto-ontology represents the starting point for the development of a Software Engineering domain ontology: it is based on an already consensual domain knowledge (e.g. the SWEBOK Guide) and will serve as an initial focus to the domain experts starting up the ontology construction process;

- The descriptions contained in the SWEBOK Guide were analysed and the concepts, relationships between concepts, terms and definitions were extracted, one SWEBOK knowledge area at a time;

- Some definitions for the concepts extracted were complemented using the 610.12-1190 IEEE Standard Glossary (1200+ entries);
- Output from term extraction tools (UQAM-LANCI's NUMEXCO) are used to ensure completeness of the proto-ontology concepts.

The Ontology Development Process

The Internal V&E cycle:

- Internal validation cycle : ETS – UQAM;
- Goal:
  Initial validation about the elements (concepts, attributes, properties and relationships) contained in the software engineering proto-ontology.

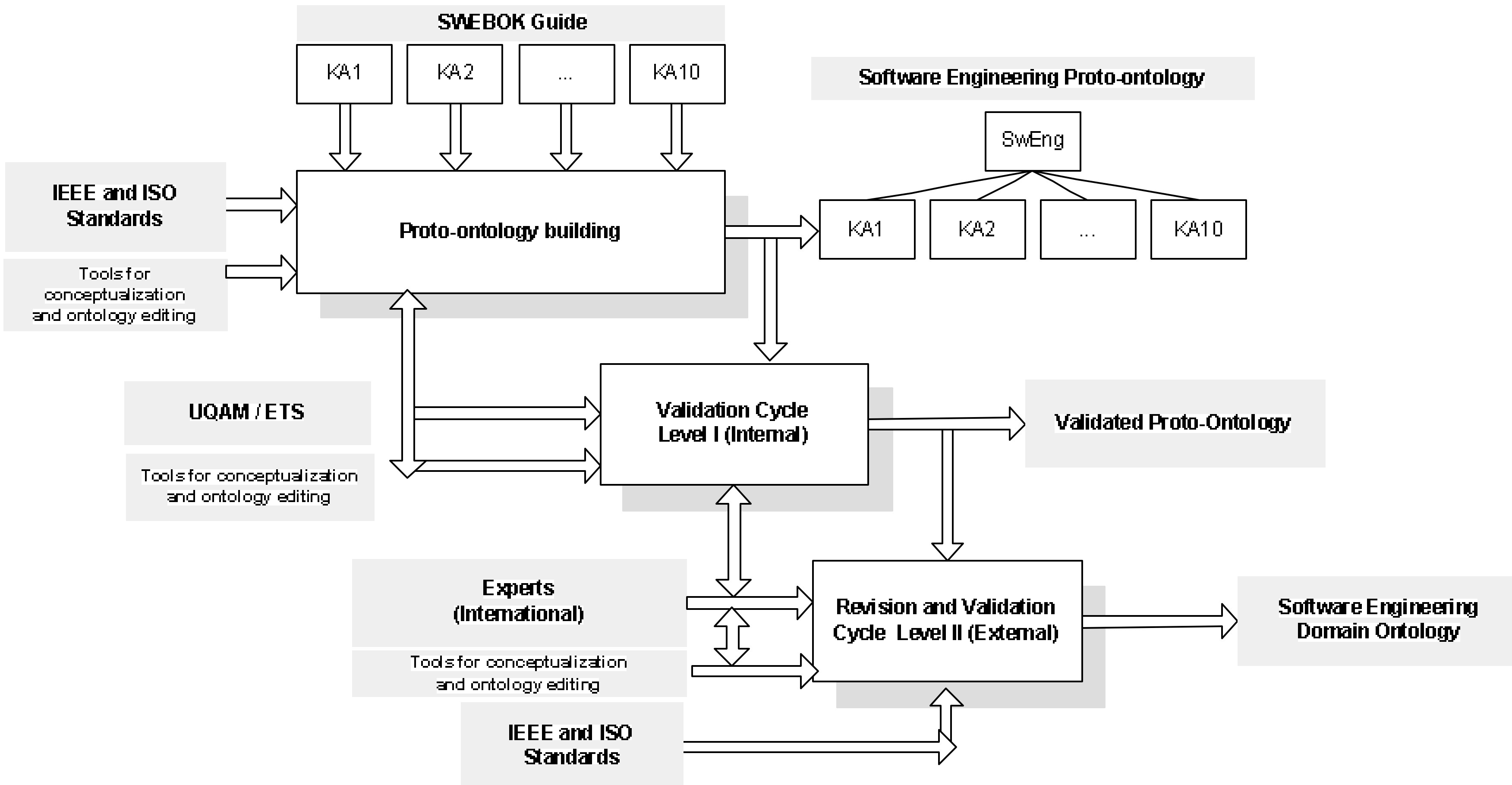
The Ontology Development Process

(5 / 5)

The <u>External</u> V&E cycle:

- A series of external proto-ontology validation and extension (V&E) cycles was started in June, 2005.
- Goal:
  Aided by panels of international software engineering domain experts, to build progressively larger consensus about the concepts, attributes and relationships that should be present in the final software engineering ontology.
- The V&E phase is performed on the *conceptual level* of the SWEBOK proto-ontology.
- Once completed the V&E cycle, the SWEBOK ontology is translated to the operational level using the OWL language and an ontology editor.

The V&E Strategy (1/2)



The V&E Strategy (2/2)

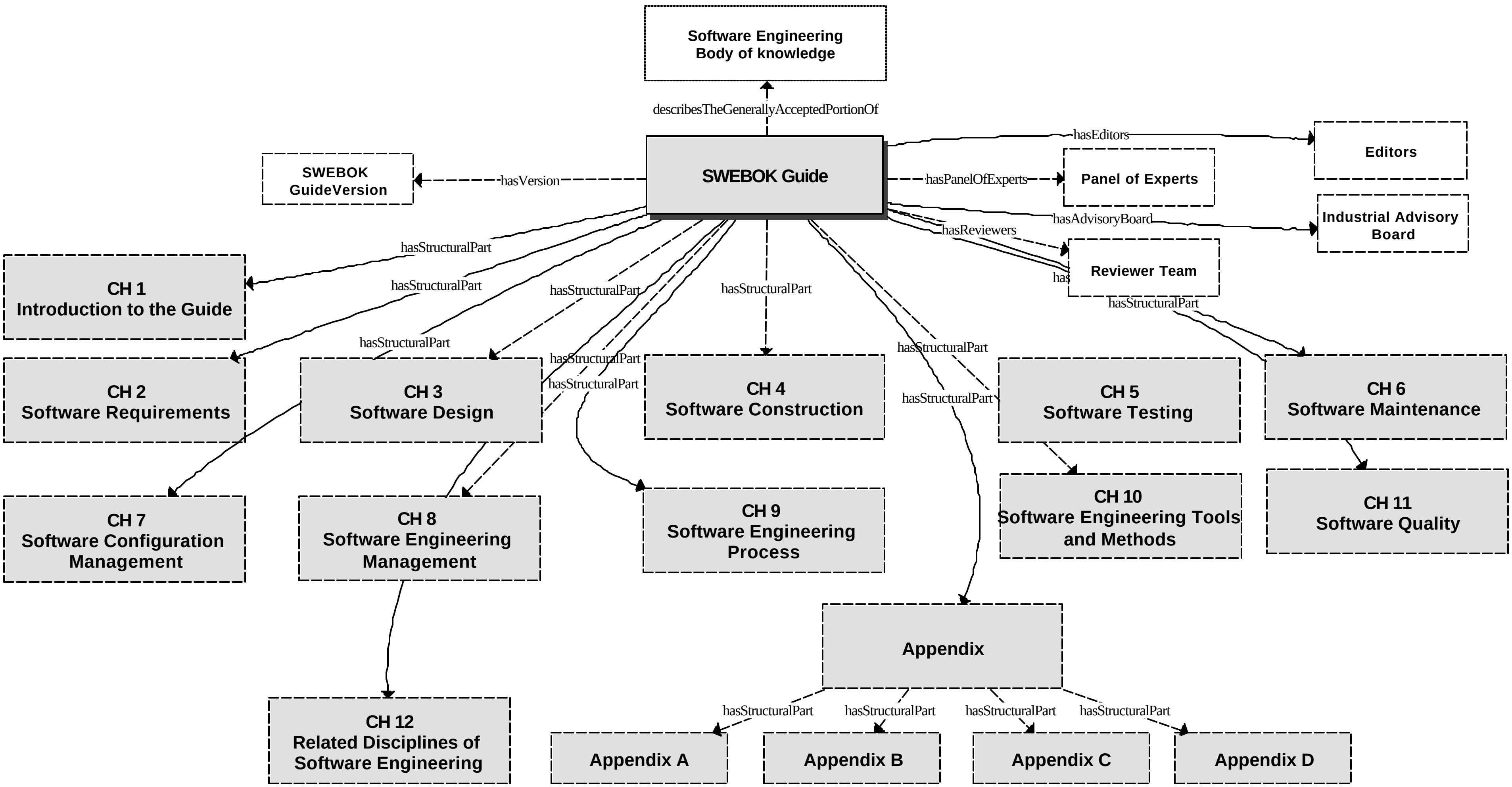
- **Inputs**
  - SWEBOK Guide; Technical Standards (IEEE, ISO)
  - Proto-ontology conceptual level;
  - Participants : Domain experts (4+), Proto-ontology developer;
- **Outputs**
  - Document recording the proposed modifications to concepts or relationships;
  - Proto-ontology : Validated and Extended;
- **Duration of the V&E sessions:** 4 hours.
- **Planned 2005 Summer V&E sessions :**
  - Internal (UQAM – ETS) : 1 (pre-evaluation);
  - Regional : 3+;
  - International : 2+.

Results (1/4)

- **The proto-ontology development phase has identified based on the SWEBOK Guide:**
  - **Concepts: over 6,000**;
  - **Normalized relationships** (to limit and standardize the great variety of terms)
  - **Facts:** over1,200 facts (examples/instances of concepts)
  - **Index:** represents the structure of the SWEBOK guide (permit to trace back where a concept is used in the guide)

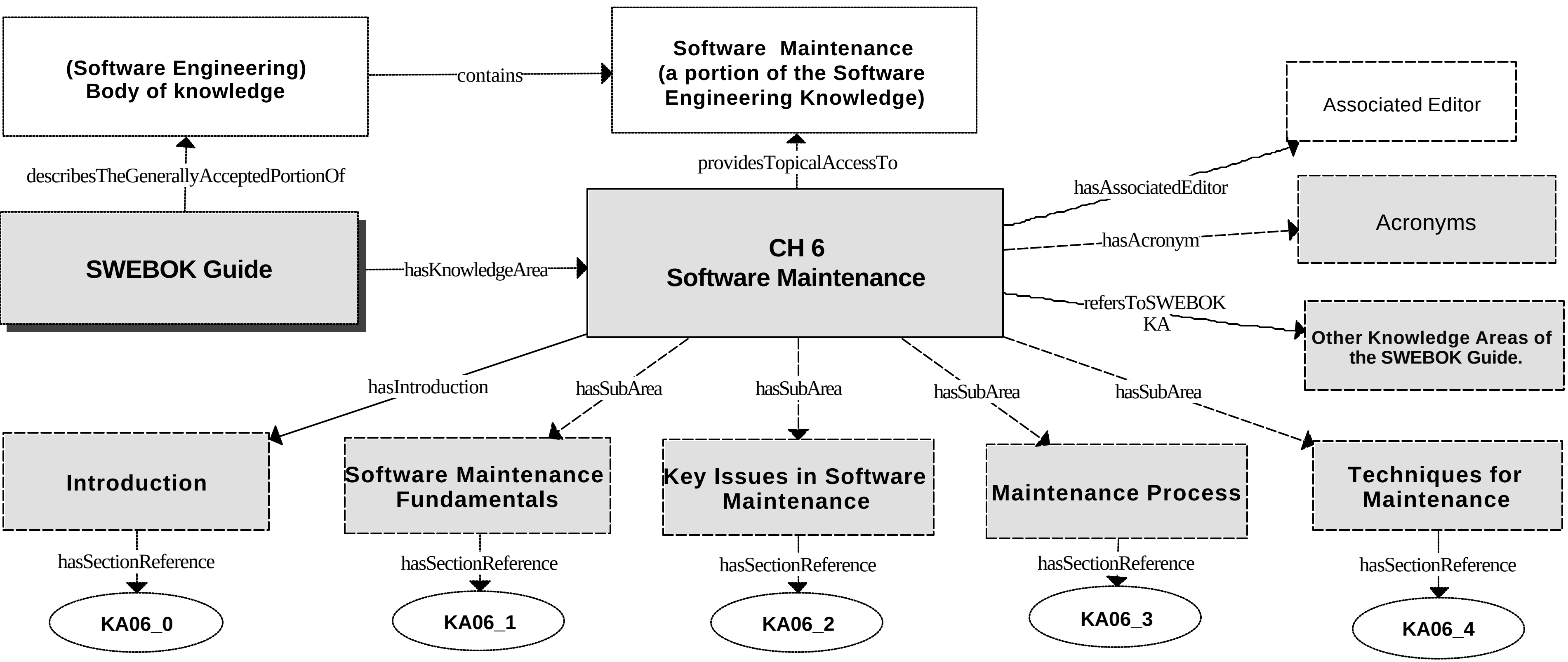
	Relationships	Index	Concepts	Facts
SWEBOK (Main structure)	6	0	**39**	57
KA 01 Introduction	25	0	**673**	14
KA 02 Software Requirements	41	44	**205**	72
KA 03 Software Design	46	45	**267**	200
KA 04 Software Construction	23	20	**200**	62
KA 05 Software Testing	97	101	**1048**	165
KA 06 Software Maintenance	47	45	**725**	141
KA 07 Software Configuration Management	51	56	**960**	102
KA 08 Software Engineering Management	40	38	**1059**	109
KA 09 Software Engineering Process	45	37	**562**	134
KA 10 Software Engineering Tools and Methods	19	51	**198**	58
KA 11 Software Quality	37	34	**412**	82
CH 12 Related Disciplines of Software Engineering	12	0	**164**	32
TOTAL		**471**	**6512**	**1228**

Results (2/4)

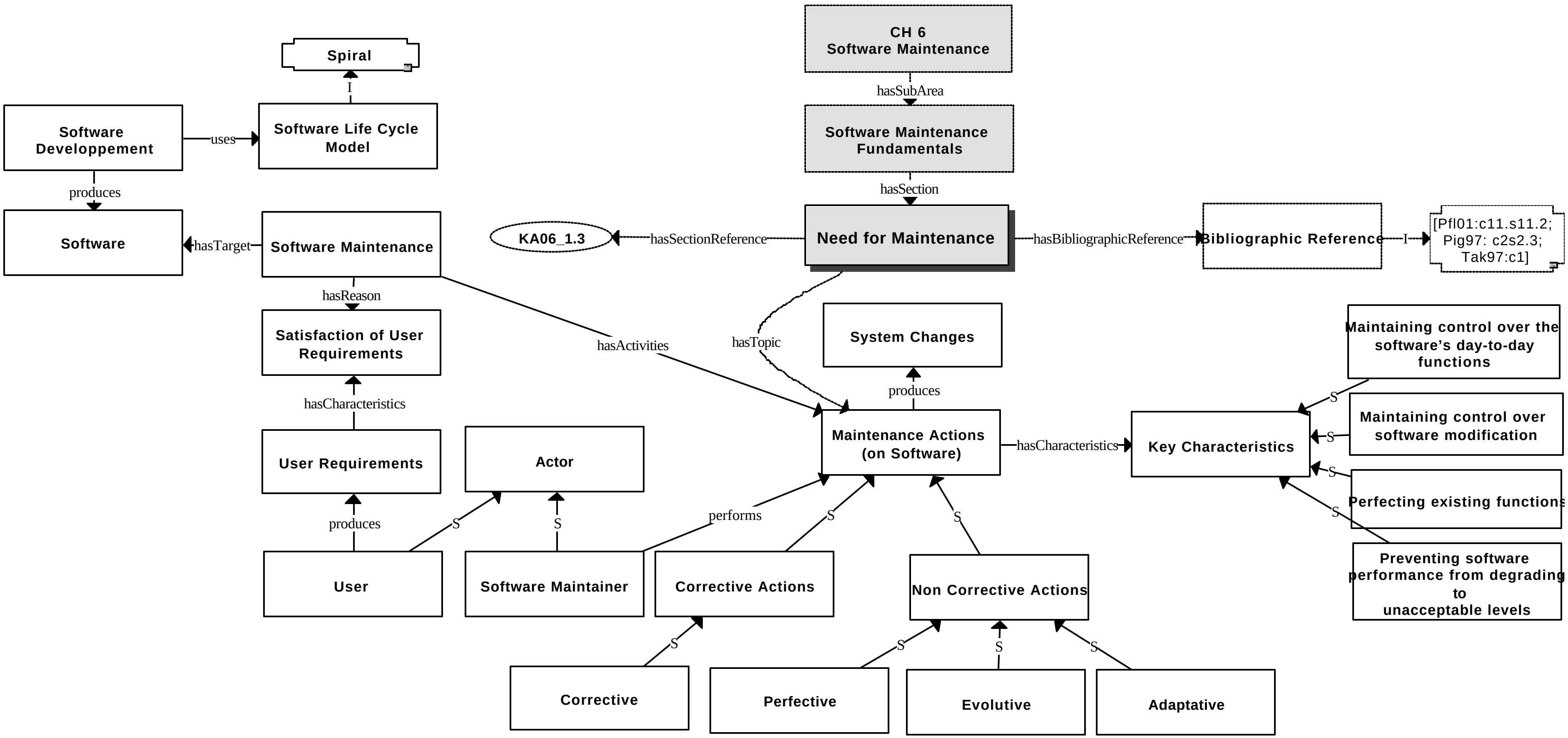


Software Engineering Body of knowledge
describesTheGenerallyAcceptedPortionOf
SWEBOK Guide
hasEditors
Editors
SWEBOK GuideVersion
hasVersion
hasPanelOfExperts
Panel of Experts
hasAdvisoryBoard
Industrial Advisory Board
hasReviewers
Reviewer Team
has
hasStructuralPart
CH 1 Introduction to the Guide
CH 2 Software Requirements
CH 3 Software Design
CH 4 Software Construction
CH 5 Software Testing
CH 6 Software Maintenance
CH 7 Software Configuration Management
CH 8 Software Engineering Management
CH 9 Software Engineering Process
CH 10 Software Engineering Tools and Methods
CH 11 Software Quality
CH 12 Related Disciplines of Software Engineering
Appendix
Appendix A
Appendix B
Appendix C
Appendix D

Results (3 / 4)

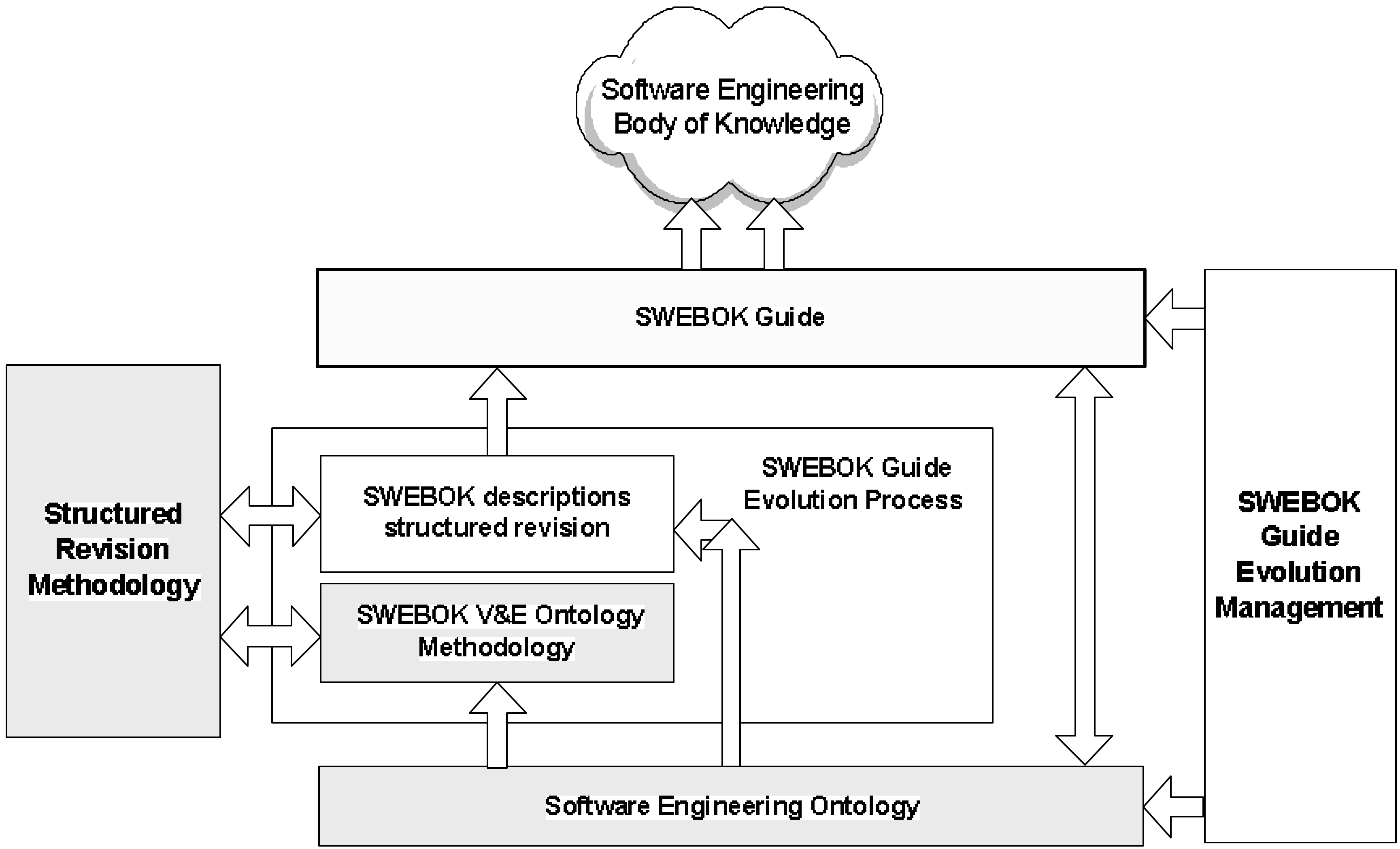

(Software Engineering) Body of knowledge
contains
Software Maintenance (a portion of the Software Engineering Knowledge)
describesTheGenerallyAcceptedPortionOf
providesTopicalAccessTo
Associated Editor
hasAssociatedEditor
Acronyms
hasAcronym
SWEBOK Guide
hasKnowledgeArea
CH 6 Software Maintenance
refersToSWEBOK KA
Other Knowledge Areas of the SWEBOK Guide.
hasIntroduction
hasSubArea
hasSubArea
hasSubArea
hasSubArea
Introduction
Software Maintenance Fundamentals
Key Issues in Software Maintenance
Maintenance Process
Techniques for Maintenance
hasSectionReference
hasSectionReference
hasSectionReference
hasSectionReference
hasSectionReference
KA06_0
KA06_1
KA06_2
KA06_3
KA06_4

Results (4 / 4)

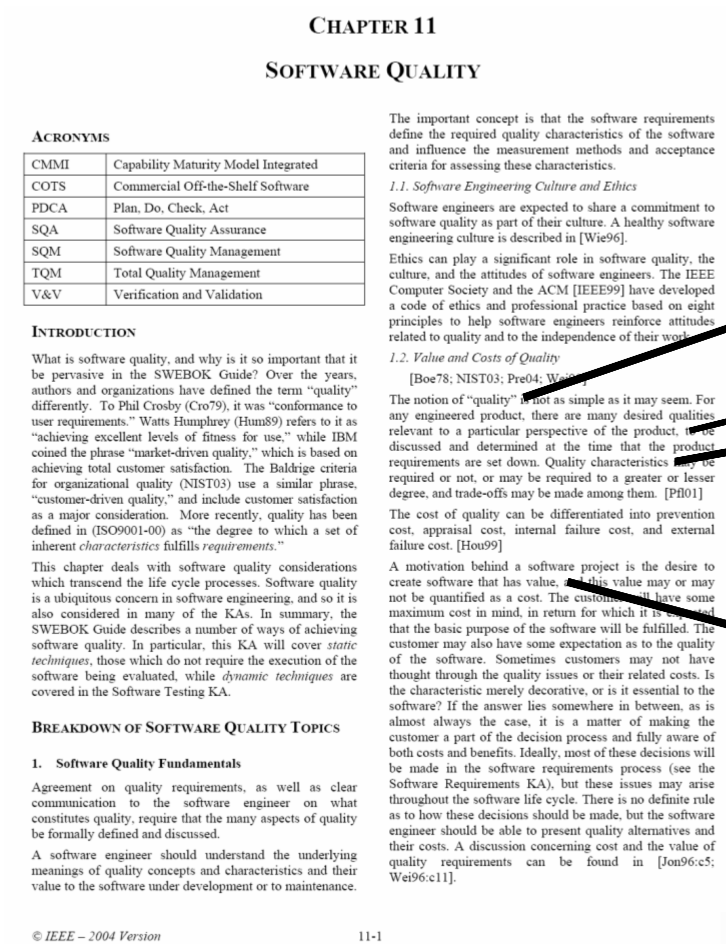


Spiral
I
Software Developpement
uses
Software Life Cycle Model
produces
Software
hasTarget
Software Maintenance
hasReason
Satisfaction of User Requirements
hasCharacteristics
User Requirements
produces
User
S
Actor
S
Software Maintainer
performs
hasActivities
CH 6 Software Maintenance
hasSubArea
Software Maintenance Fundamentals
hasSection
KA06_1.3
hasSectionReference
Need for Maintenance
hasBibliographicReference
Bibliographic Reference
I
[Pfl01:c11.s11.2; Pig97: c2s2.3; Tak97:c1]
hasTopic
System Changes
produces
Maintenance Actions (on Software)
hasCharacteristics
Key Characteristics
Maintaining control over the software's day-to-day functions
Maintaining control over software modification
Perfecting existing functions
Preventing software performance from degrading to unacceptable levels
Corrective Actions
Non Corrective Actions
Corrective
Perfective
Evolutive
Adaptative

Sw. Eng. Ontology: Application



Ontology based structured revision
(1 / 3)



CHAPTER 11

SOFTWARE QUALITY

ACRONYMS

CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-the-Shelf Software
PDCA	Plan, Do, Check, Act
SQA	Software Quality Assurance
SQM	Software Quality Management
TQM	Total Quality Management
V&V	Verification and Validation

INTRODUCTION

What is software quality, and why is it so important that it be pervasive in the SWEBOK Guide? Over the years, authors and organizations have defined the term “quality” differently. To Phil Crosby (Cro79), it was “conformance to user requirements.” Watts Humphrey (Hum89) refers to it as “achieving excellent levels of fitness for use,” while IBM coined the phrase “market-driven quality,” which is based on achieving total customer satisfaction. The Baldrige criteria for organizational quality (NIST03) use a similar phrase, “customer-driven quality,” and include customer satisfaction as a major consideration. More recently, quality has been defined in (ISO9001-00) as “the degree to which a set of inherent *characteristics* fulfills *requirements*.”

This chapter deals with software quality considerations which transcend the life cycle processes. Software quality is a ubiquitous concern in software engineering, and so it is also considered in many of the KAs. In summary, the SWEBOK Guide describes a number of ways of achieving software quality. In particular, this KA will cover *static techniques*, those which do not require the execution of the software being evaluated, while *dynamic techniques* are covered in the Software Testing KA.

BREAKDOWN OF SOFTWARE QUALITY TOPICS

1. **Software Quality Fundamentals**

Agreement on quality requirements, as well as clear communication to the software engineer on what constitutes quality, require that the many aspects of quality be formally defined and discussed.

A software engineer should understand the underlying meanings of quality concepts and characteristics and their value to the software under development or to maintenance.

The important concept is that the software requirements define the required quality characteristics of the software and influence the measurement methods and acceptance criteria for assessing these characteristics.

1.1. Software Engineering Culture and Ethics

Software engineers are expected to share a commitment to software quality as part of their culture. A healthy software engineering culture is described in [Wie96].

Ethics can play a significant role in software quality, the culture, and the attitudes of software engineers. The IEEE Computer Society and the ACM [IEEE99] have developed a code of ethics and professional practice based on eight principles to help software engineers reinforce attitudes related to quality and to the independence of their work.

1.2. Value and Costs of Quality

[Boe78; NIST03; Pre04; Wei96]

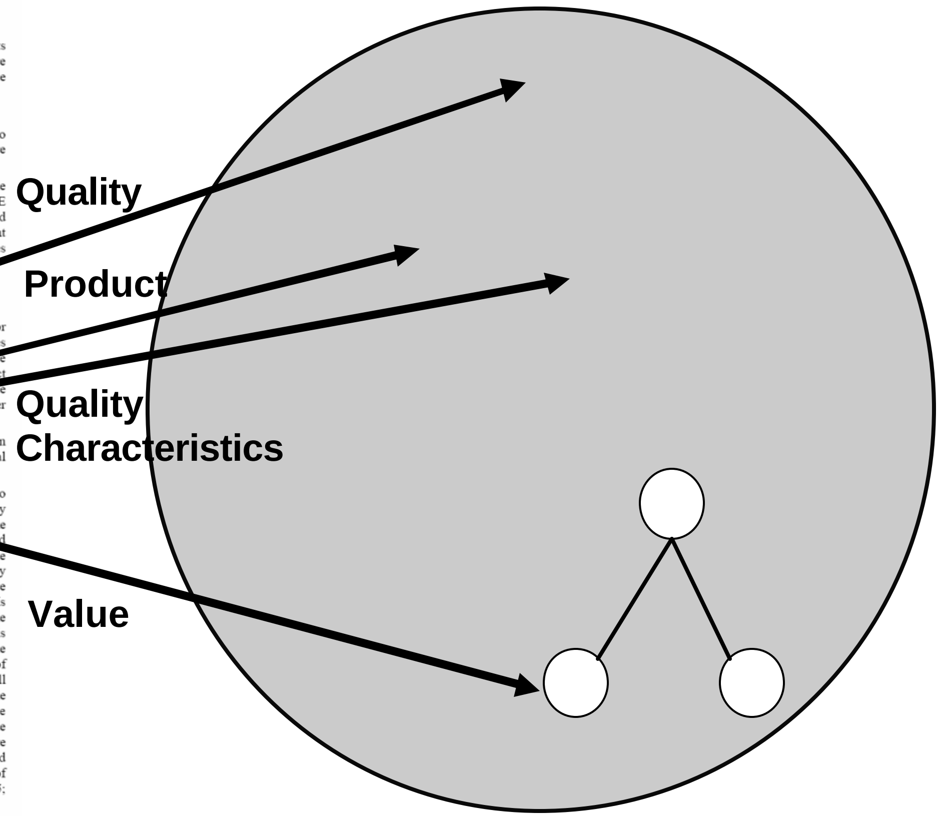
The notion of “quality” is not as simple as it may seem. For any engineered product, there are many desired qualities relevant to a particular perspective of the product, to be discussed and determined at the time that the product requirements are set down. Quality characteristics may be required or not, or may be required to a greater or lesser degree, and trade-offs may be made among them. [Pfl01]

The cost of quality can be differentiated into prevention cost, appraisal cost, internal failure cost, and external failure cost. [Hou99]

A motivation behind a software project is the desire to create software that has value, and this value may or may not be quantified as a cost. The customer will have some maximum cost in mind, in return for which it is expected that the basic purpose of the software will be fulfilled. The customer may also have some expectation as to the quality of the software. Sometimes customers may not have thought through the quality issues or their related costs. Is the characteristic merely decorative, or is it essential to the software? If the answer lies somewhere in between, as is almost always the case, it is a matter of making the customer a part of the decision process and fully aware of both costs and benefits. Ideally, most of these decisions will be made in the software requirements process (see the Software Requirements KA), but these issues may arise throughout the software life cycle. There is no definite rule as to how these decisions should be made, but the software engineer should be able to present quality alternatives and their costs. A discussion concerning cost and the value of quality requirements can be found in [Jon96:c5; Wei96:c11].

© IEEE – 2004 Version 11-1

Software Engineering Ontology

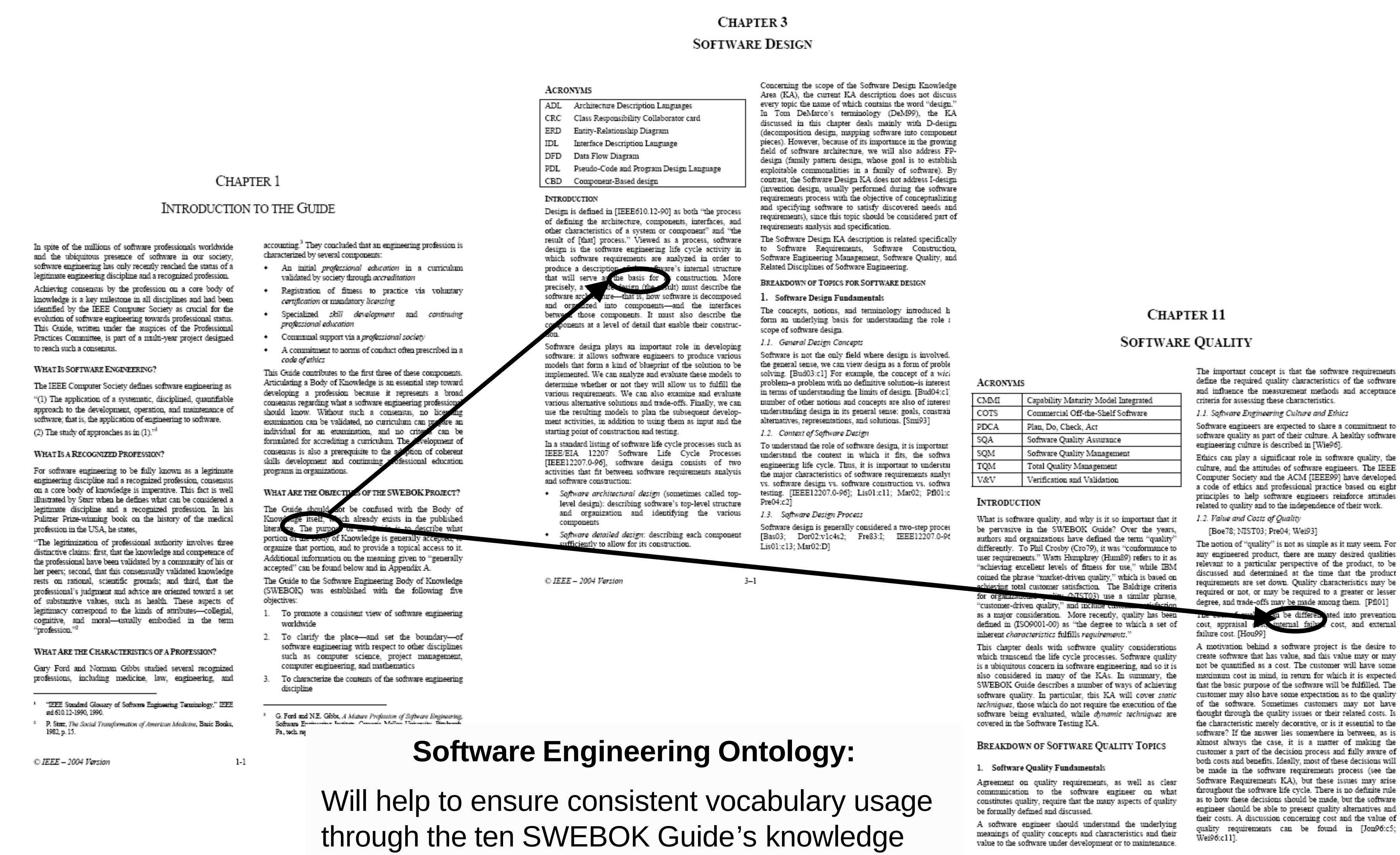


SWEBOK Guide’s <u>structured</u> <u>revision</u> will ensure:

- **Vocabulary usage harmonization**
- **Descriptions level of detail harmonization**

Ontology based structured revision

(2 / 3)



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 only recently reached the status of a legitimate engineering discipline and a recognized profession.

Achieving consensus by the profession on a core body of knowledge is a key milestone in all disciplines and had been identified by the IEEE Computer Society as crucial for the evolution of software engineering towards professional status. This Guide, written under the auspices of the Professional Practices Committee, is part of a multi-year project designed to reach such a 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 fully 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,

"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 embodied in the term "profession."[2]

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 consensus is also a prerequisite to the adoption of coherent skills 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, which already exists in the published literature. The purpose of the Guide is to describe what portion of the Body of Knowledge is generally accepted, to organize that portion, and to provide a topical access to it. Additional information on the meaning given to "generally accepted" can be found below and in Appendix A.

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

1. To promote a consistent view of software engineering worldwide
2. To 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. To characterize the contents of the software engineering discipline

[1] "IEEE Standard Glossary of Software Engineering Terminology," IEEE 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, Pa., tech. rep.

© IEEE – 2004 Version 1-1

CHAPTER 3

SOFTWARE DESIGN

ACRONYMS

ADL	Architecture Description Languages
CRC	Class Responsibility Collaborator card
ERD	Entity-Relationship Diagram
IDL	Interface Description Language
DFD	Data Flow Diagram
PDL	Pseudo-Code and Program Design Language
CBD	Component-Based design

INTRODUCTION

Design is defined in [IEEE610.12-90] as both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process." Viewed as a process, software design is the software engineering life cycle activity in which software requirements are analyzed in order to produce a description of the software's internal structure that will serve as the basis for its construction. More precisely, a software design (the result) must describe the software architecture—that is, how software is decomposed and organized into components—and the interfaces between those components. It must also describe the components at a level of detail that enable their construction.

Software design plays an important role in developing software: it allows software engineers to produce various models that form a kind of blueprint of the solution to be implemented. We can analyze and evaluate these models to determine whether or not they will allow us to fulfill the various requirements. We can also examine and evaluate various alternative solutions and trade-offs. Finally, we can use the resulting models to plan the subsequent development activities, in addition to using them as input and the starting point of construction and testing.

In a standard listing of software life cycle processes such as IEEE/EIA 12207 Software Life Cycle Processes [IEEE12207.0-96], software design consists of two activities that fit between software requirements analysis and software construction:

- *Software architectural design* (sometimes called top-level design): describing software's top-level structure and organization and identifying the various components
- *Software detailed design*: describing each component sufficiently to allow for its construction.

Concerning the scope of the Software Design Knowledge Area (KA), the current KA description does not discuss every topic the name of which contains the word "design." In Tom DeMarco's terminology (DeM99), the KA discussed in this chapter deals mainly with D-design (decomposition design, mapping software into component pieces). However, because of its importance in the growing field of software architecture, we will also address FP-design (family pattern design, whose goal is to establish exploitable commonalities in a family of software). By contrast, the Software Design KA does not address I-design (invention design, usually performed during the software requirements process with the objective of conceptualizing and specifying software to satisfy discovered needs and requirements), since this topic should be considered part of requirements analysis and specification.

The Software Design KA description is related specifically to Software Requirements, Software Construction, Software Engineering Management, Software Quality, and Related Disciplines of Software Engineering.

BREAKDOWN OF TOPICS FOR SOFTWARE DESIGN

1. Software Design Fundamentals

The concepts, notions, and terminology introduced h form an underlying basis for understanding the role scope of software design.

1.1. General Design Concepts

Software is not the only field where design is involved. the general sense, we can view design as a form of proble solving. [Bud03:c1] For example, the concept of a wic problem–a problem with no definitive solution–is interest in terms of understanding the limits of design. [Bud04:c1] number of other notions and concepts are also of interest understanding design in its general sense: goals, constrai alternatives, representations, and solutions. [Smi93]

1.2. Context of Software Design

To understand the role of software design, it is important understand the context in which it fits, the softwa engineering life cycle. Thus, it is important to understa the major characteristics of software requirements analys vs. software design vs. software construction vs. softwa testing. [IEEE12207.0-96]; Lis01:c11; Mar02; Pfl01:c Pre04:c2]

1.3. Software Design Process

Software design is generally considered a two-step proces [Bas03; Dor02:v1c4s2; Fre83:I; IEEE12207.0-96 Lis01:c13; Mar02:D]

© IEEE – 2004 Version 3–1

CHAPTER 11

SOFTWARE QUALITY

ACRONYMS

CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-the-Shelf Software
PDCA	Plan, Do, Check, Act
SQA	Software Quality Assurance
SQM	Software Quality Management
TQM	Total Quality Management
V&V	Verification and Validation

INTRODUCTION

What is software quality, and why is it so important that it be pervasive in the SWEBOK Guide? Over the years, authors and organizations have defined the term "quality" differently. To Phil Crosby (Cro79), it was "conformance to user requirements." Watts Humphrey (Hum89) refers to it as "achieving excellent levels of fitness for use," while IBM coined the phrase "market-driven quality," which is based on achieving total customer satisfaction. The Baldrige criteria for organizational quality (NIST03) use a similar phrase, "customer-driven quality," and include customer satisfaction as a major consideration. More recently, quality has been defined in (ISO9001-00) as "the degree to which a set of inherent *characteristics* fulfills *requirements*."

This chapter deals with software quality considerations which transcend the life cycle processes. Software quality is a ubiquitous concern in software engineering, and so it is also considered in many of the KAs. In summary, the SWEBOK Guide describes a number of ways of achieving software quality. In particular, this KA will cover *static techniques*, those which do not require the execution of the software being evaluated, while *dynamic techniques* are covered in the Software Testing KA.

BREAKDOWN OF SOFTWARE QUALITY TOPICS

1. Software Quality Fundamentals

Agreement on quality requirements, as well as clear communication to the software engineer on what constitutes quality, require that the many aspects of quality be formally defined and discussed.

A software engineer should understand the underlying meanings of quality concepts and characteristics and their value to the software under development or to maintenance.

The important concept is that the software requirements define the required quality characteristics of the software and influence the measurement methods and acceptance criteria for assessing these characteristics.

1.1. Software Engineering Culture and Ethics

Software engineers are expected to share a commitment to software quality as part of their culture. A healthy software engineering culture is described in [Wie96].

Ethics can play a significant role in software quality, the culture, and the attitudes of software engineers. The IEEE Computer Society and the ACM [IEEE99] have developed a code of ethics and professional practice based on eight principles to help software engineers reinforce attitudes related to quality and to the independence of their work.

1.2. Value and Costs of Quality

[Boe78; NIST03; Pre04; Wei93]

The notion of "quality" is not as simple as it may seem. For any engineered product, there are many desired qualities relevant to a particular perspective of the product, to be discussed and determined at the time that the product requirements are set down. Quality characteristics may be required or not, or may be required to a greater or lesser degree, and trade-offs may be made among them. [Pfl01]

The cost of quality can be differentiated into prevention cost, appraisal cost, internal failure cost, and external failure cost. [Hou99]

A motivation behind a software project is the desire to create software that has value, and this value may or may not be quantified as a cost. The customer will have some maximum cost in mind, in return for which it is expected that the basic purpose of the software will be fulfilled. The customer may also have some expectation as to the quality of the software. Sometimes customers may not have thought through the quality issues or their related costs. Is the characteristic merely decorative, or is it essential to the software? If the answer lies somewhere in between, as is almost always the case, it is a matter of making the customer a part of the decision process and fully aware of both costs and benefits. Ideally, most of these decisions will be made in the software requirements process (see the Software Requirements KA), but these issues may arise throughout the software life cycle. There is no definite rule as to how these decisions should be made, but the software engineer should be able to present quality alternatives and their costs. A discussion concerning cost and the value of quality requirements can be found in [Jon96:c5; Wei96:c11].

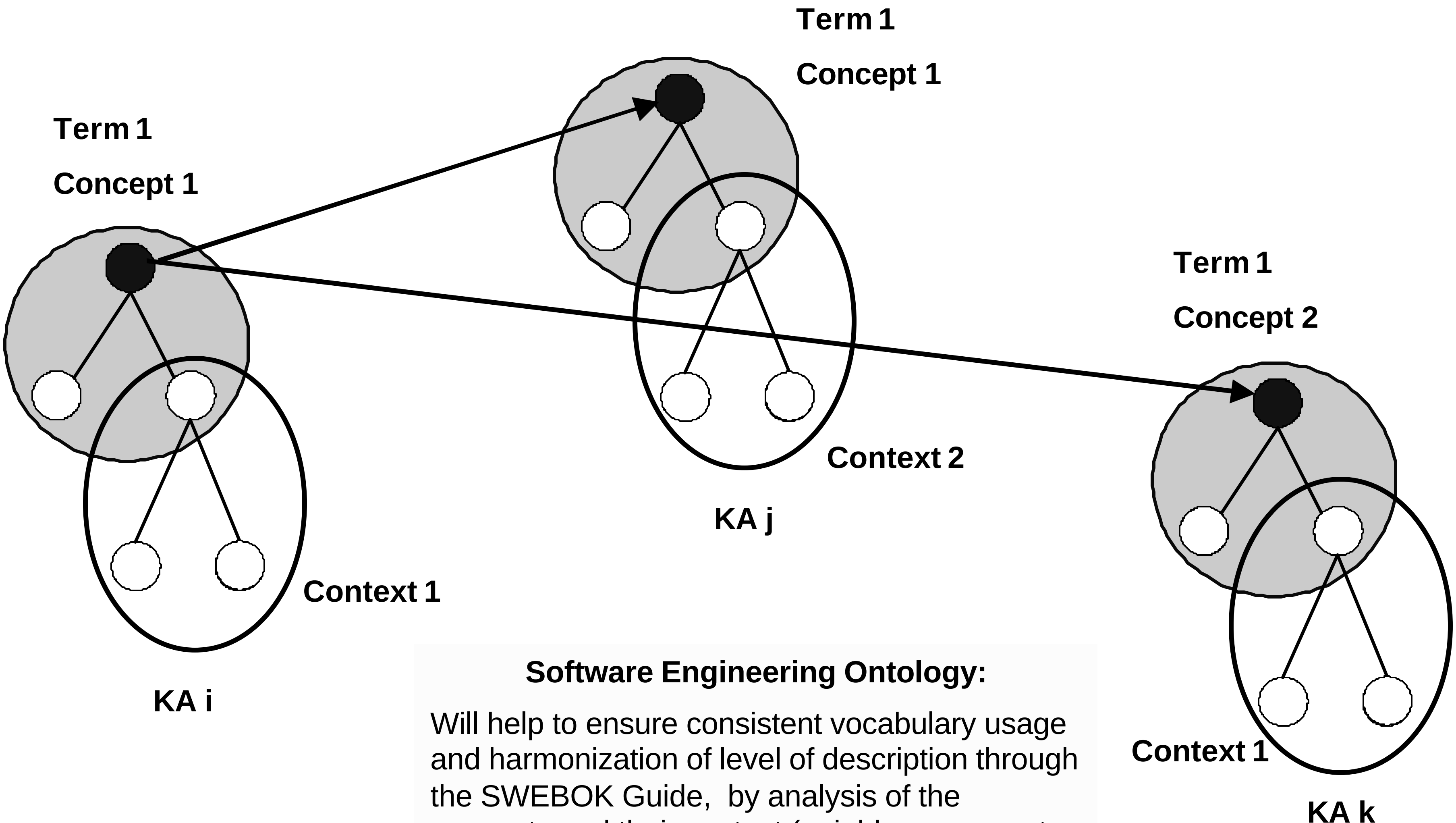
© IEEE – 2004 Version 11-1

Software Engineering Ontology:

Will help to ensure consistent vocabulary usage through the ten SWEBOK Guide's knowledge areas (KAs)

Ontology based structured revision

(3 / 3)



Software Engineering Ontology:

Will help to ensure consistent vocabulary usage and harmonization of level of description through the SWEBOK Guide, by analysis of the concepts and their context (neighbors concepts and relationships), supported by the software engineering ontology

Summary

- Our project goal is to build and validate an ontology for the Software Engineering discipline.
- To reach this goal, an initial domain ontology (e.g. a proto-ontology) was developed for the software engineering area, taking as starting point the consensual knowledge already acquired, structured, validated and made available by the SWEBOK project (SWEBOK Guide - Iron Man version, 18.05.2004).
- Technical standards (IEEE and ISO) will also be used to complete the Software Engineering Ontology, providing for definitions of the currently accepted terminology as well as alternate accepted terms.
- The resulting domain ontology will integrate a set of artefacts corresponding to the conceptual, ontological and operational levels of the software engineering validated ontology.

Research Contributions (1/2)

- Identification of the main inputs, outputs and activities to be performed in order to develop the domain Software Engineering ontology.

- Identification of the main software engineering concepts, terms, definitions, relationships between concepts (IsA, PartOf, and other specifics relationships) and axioms describing the concepts.

- Domain expert validation of the Software Engineering Ontology.

- Progressive building of a consensus concerning the concepts in the ontology aided by international software engineering domain experts.

Research Contributions (2/2)

- The use of this “Software Engineering Ontology” may also contribute later to the development of additional content validation by carrying out *automatic* cross-correlation validation across the ten areas of knowledge in the SWEBOK Guide.

- This next step would ensure that all concepts and definitions are used in a consistent fashion throughout all ten SWEBOK knowledge areas as well as to harmonize the level of description of the SWEBOK Guide content.

- An automatic validation would also be useful in ISO, contributing to **ISO/IEC JTC1 SC-7/SWG5** efforts towards the re-synchronisation of *software engineering* technical standards and harmonization of all vocabulary used by the various ISO software engineering working groups.

Future Work

Further work in this project will include:

- Completion of the SWEBOK Ontology V&E cycles
  The validation and extension (V&E) cycles with panels of domain experts will produce a series of sub-ontologies that, once integrated and operationalized in OWL, will form the SWEBOK ontology.

- Cognition-communication analysis
  To observe and analyse the interactions that take place among the group of domain experts when they are working collaboratively to validate and extend the SWEBOK proto-ontology.

  Description and modelling the communication interactions and the cognitive activities that emerge within the *distributed cognitive system* formed by the experts working in the V&E of the SWEBOK ontology.

  This will contribute to:
  - Identify major key issues and challenges in the ontology V&E process;
  - To formulate some recommendations aiming at improving the global efficiency of the ontology construction process.

Bibliographic References

[1] Porphyry “Isagoge”, Vrin, 1998, ISBN: 2711613445

[2] Patil et al. 1992, “The DARPA Knowledge Sharing Effort: Progress Report”, Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference KR'92, San Mateo, California, p: 777-788.

[2] Neeches R., F. R. E., Finin T., Gruber T. R., Senator T., and Swartout W. R., 1991. “Enabling technology for knowledge sharing ”, AI Magazine, 12, p: 35-56.

[4] Davenport, Thomas H., 1993, “Process Innovation: Reengineering Work Through Information Technology”, Boston, MA, Harvard Business School Press.

[5] Guarino, N., Schneider , L., 2002, “Ontology-Driven Conceptual Modelling”, Lecture Notes In Computer Science; Vol. 2503 Proceedings of the 21st International Conference on Conceptual Modeling, ISBN:3-540-44277

[6] Gruber, T.R.,. 1993, “Towards Principles for the Design of Ontologies Used for Knowledge Sharing”, in Roberto Poli Nicola Guarino, editor, International Workshop on Formal Ontology, Padova, Italy, 1993, Technical report KSL-93-04, Knowledge Systems Laboratory, Stanford University.

[7] Rector, A., Schreiber, G., Noy, N. F., Knublauch H. and Musen, M., 2004, “Ontology Design Patterns and Problems ”, Tutorial at the Third International Semantic Web Conference (ISWC 2004), November 7th, 2004.

[8] Gruninger, M., Lee, Jintae, 2002, “Ontology Design and Applications”. Communications of the ACM, February 2002, 45 (2), p: 1-2.

[9] Wille, C., Abran, A., Desharnais, J.M.., Dumke, R., 2003, “The Quality concepts and sub concepts in SWEBOK: An ontology challenge”, in International Workshop on Software Measurement (IWSM), Montreal, 2003 , p. 18.

[10] Wille, C., Dumke, R., Abran, A., Desharnais, JM. ,2004. E-Learning Infrastructure for Software Engineering Education: Steps in Ontology Modeling for SWEBOK, Software Measurement European Forum, Rome, Italy.

[11] Mendes, O., Abran, A. 2004. “Software Engineering Ontology: A Development Methodology”, Position Paper, Metrics News 9:1,August, p: 68-76

[12] Bourque, P., Dupuis, R., Abran, A., 1999, “The Guide to the Software Engineering Body of Knowledge”, IEEE Software, November/December.

[13] Abran, A., Moore, J., Bourque, P., Dupuis, R., Tripp, L., Guide to the Software Engineering Body of Knowledge – SWEBOK, Iron Man Version 1.0, IEEE-Computer Society Press, to be published 2005, URL: http://www.swebok.org

[14] Kitchenham, B., et al. 1999, “Towards a software maintenance ontology”, Journal of Software Maintenance: Research and Practice, Vol. 11, p: 365-389.

[15] Ruiz, F., Vizcaíno, A., Piattini, M. y García, F., 2004, “An Ontology for the Management of Software Maintenance Projects”, International Journal of Software Engineering and Knowledge Engineering, Vol. 14, No. 3, p: 323-349.

[16] Martin, M de A., Olsina, L., 2003, “Towards an Ontology for Software Metrics and Indicators as the Foundation for a Cataloging Web System ”, First Latin American Web Congress (LA-WEB'03). 10 - 11, 2003. Santiago, Chile.

[17] Garzás J., Piattini M. 2005, “An Ontology for Microarchitectural Design Knowledge”, IEEE Software Vol. 29, p: 28 -33.

[18] Mendes, O., 2004, “Méthodologies de construction d’ontologies”, Congrès de l’ACFAS, Montreal May 12

Université du Québec à Montréal

Université du Québec
École de technologie supérieure

Thank You

For your attention

UQÀM
Université du Québec à Montréal

Université du Québec
École de technologie supérieure

For additional questions

Olavo Mendes olavomendes@gmail.com

Alain Abran abran.alain@ets.ca