Everything is Possible to Structure – Even the Software Engineering Body of Knowledge

Mira Kajko-Mattsson, Anders Sjögren, Leif Lindbäck
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden
mekm2@kth.se, as@kth.se, leifl@kth.se

***Abstract*—Everything is possible to structure, even the software engineering body of knowledge. In this paper, we suggest a conceptual model of the software engineering body of knowledge. The model is a restructured version of SWEBOK and ACM/IEEE Curriculum Guidelines. It constitutes the first attempt to create an underlying structure that is common to most of the software engineering bodies of knowledge.**

Keywords-conceptual model of the software engineering body of knowledge; SWEBOK, knowledge areas; knowledge subareas.

I. Introduction

Development of a software engineering body of knowledge is a formidable undertaking [2][3]. It has to meet the challenge of rapidly changing landscape of software engineering [2] and the challenge of accommodating to the diversity of the current and new emerging domains that strongly rely on software engineering [2][3]. Such a body of knowledge should guide in developing software engineering curricula. It should constitute a basis for evaluating the knowledge and skills of software graduates and professionals and it should provide a roadmap for following up their lifelong progression [6]. This requires a solid structure of the software engineering body of knowledge that stands the test of time [6].

Today, there are few bodies of software engineering knowledge [3]. The most known one is SWEBOK - Guide to the Software Engineering Body of Knowledge [4]. Its objective is to provide foundation for curriculum development. It has been used as a foundation for developing ACM/IEEE Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering [1].

Both SWEBOK and ACM/IEEE Curriculum have for many years been developed and revised by many highly skilled professionals. Hence, they constitute a good roadmap of software engineering skills and knowledge. Despite this, we have identified a number of problems when evaluating TIDAB - an undergraduate Programme in Computer Engineering at KTH Royal Institute of technology [9]. We have found that none of the documents, neither SWEBOK nor ACM/IEEE Curriculum, have a common underlying structure of presenting their contents.

In this paper, we present the problems that we have encountered while evaluating the TIDAB programme and we suggest a conceptual model of the software engineering body of knowledge. Our goal is to provide a model enabling a coherent, stable, viable and structured description of software engineering knowledge. Because software engineering is still evolving, it is important, to have a good and structured organization of its body of knowledge so that it can be easy to analyze and evolve, and still, stand the test of time [2].

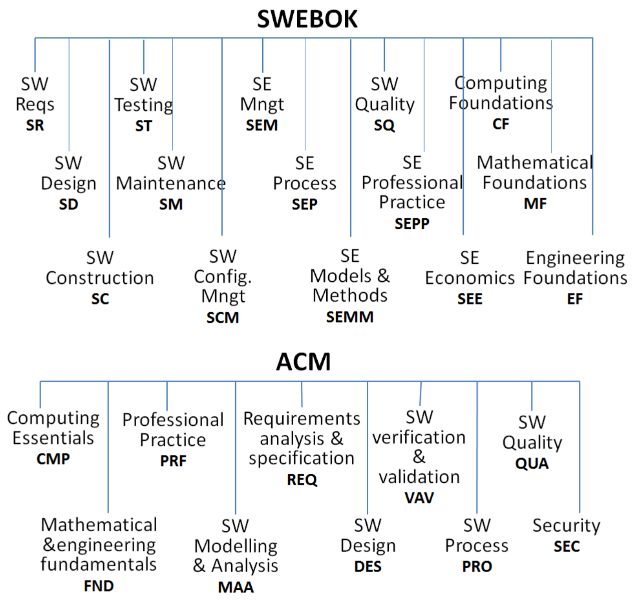


Figure 1. Design of SWEBOK and ACM/IEEE Curriculum

The remainder of this paper is as follows. Section II briefly presents some of the problems with SWEBOK and ACM Curriculum. Section III presents the conceptual model of the software engineering body of knowledge. Section IV evaluates the two standards studied. Finally, Section V makes suggestions for future work.

II. Standards Studied and Their Problems

Both SWEBOK and ACM/IEEE Curriculum include software engineering bodies of knowledge. However, both are differently wrapped and have different goals. SWEBOK provides guidance to software engineering professionals

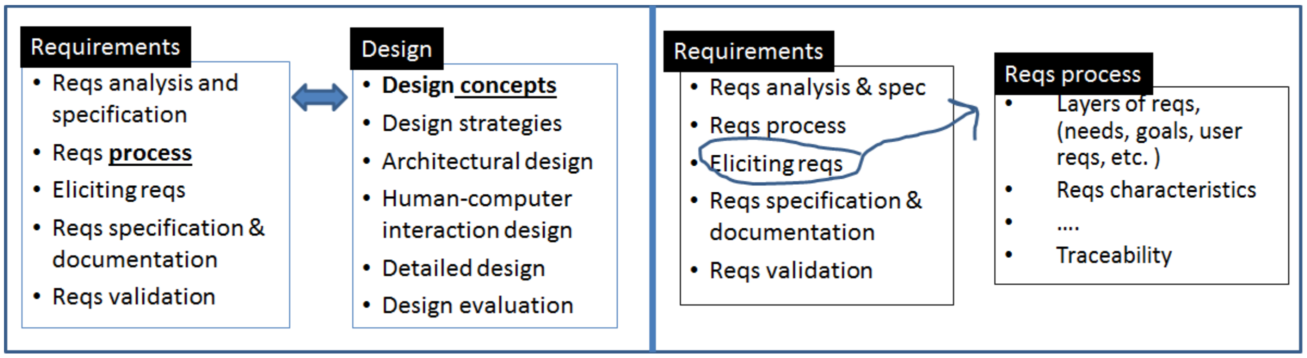


Figure 2. ACM/IEEE Curriculum's way of structuring knowledge areas

about what they should know during their lifelong career. Its overall outline is presented in the upper part of Figure 1. ACM/IEEE Curriculum, on the other hand, is based on the contents of SWEBOK and it provides guidance to academic institutions and accreditation agencies about what should constitute an undergraduate software engineering education.

The two pieces of work were originally developed by a broad, international group of highly skilled volunteer participants. Hence, the two works are of a very high quality contents-wise. Despite this, while evaluating the TIDAB programme, we have encountered some problems. These concern (1) lack of a common underlying structure for each knowledge area, (2) strange division into knowledge areas, and (3) difficulties in evaluating whether a specific Body of Knowledge (BoK) item has been fulfilled. Due to space restrictions, we only provide few examples of problems.

Regarding the first problem, we have found that the knowledge areas, be it SWEBOK or ACM/IEEE Curriculum, lack a common underlying structure. By studying Figure 2, we may see that the structure of the Requirements BoK is different than the structure of the Design BoK. Why does the Design BoK have concepts and the Requirements BoK does not and why does the Requirements BoK have process and the Design BoK does not? These are some of the questions. Similar problems have been identified in SWEBOK. To be fair to SWEBOK, however, there are some common structural elements defined in some of knowledge areas (not all) such as, for instance, *Fundamentals* and *Tools*.

We believe that all or at least the majority of the knowledge areas should have a common underlying structure. We do not know how to interpret lack of the underlying structure in the two standards studied. What we know is that the software engineering knowledge areas differ much with respect to their contents and the depth of coverage. The standards were authored by separate volunteer groups that focused on different knowledge areas. Probably, the authors were not able to reach a common consensus that would result in a common structure.

Regarding the second problem, we do not understand why ACM/IEEE Curriculum distinguishes security and puts it on the same level as other knowledge areas. By studying the lower part of Figure 1, we may see that *Security* is not part of software quality. Instead, it is put on the same level as *Quality*. As far as we understand, the standard treats *Security* as a highly functional knowledge area. This may however confuse many readers.

Finally, the third problem deals with difficulties with evaluating educational programmes. The individual BoKs in the two documents do not offer any one-to-one relationship for ticking off whether a particular BoK has been fulfilled. Let us exemplify this with ACM/IEEE Curriculum's BoK – REQ.rfd.10 stating *Requirements management (e.g., consistency management, release planning, and reuse).* If the educational programme includes release planning but not consistency management, then how can you evaluate that requirement management has been fulfilled. Similar problems have been found in SWEBOK. As an example, how do you evaluate REQ 2.4 stating *Process Quality and Improvement*, when only process quality gets implemented in the educational programme.

III. Conceptual Model of the Software Engineering Body of Knowledge

The conceptual model of the software engineering body of knowledge as presented in this section is the result of a KTH project whose goal was to evaluate the software engineering part of the TIDAB undergraduate programme [9]. Initially, we - the project members, that is, the authors of this paper, had the ambition to use ACM/IEEE Curriculum. After having tried to map TIDAB on the curriculum, the group realized that it was not easy. Hence, we chose SWEBOK to study and analyzed whether it might be useful for TIDAB. We found SWEBOK very comprehensive, however, we felt that it was not easy to match it against TIDAB either. For this reason, we decided to create our own model.

Our model consists of five groups of knowledge areas. These are presented in Figures 3-5 in various guises. To facilitate their identification, we have used numerical identifiers. The models knowledge areas are the following:

1. *Overall Management Knowledge Area*: This area includes BoKs that are necessary for managing the software organizations on a very high level. As shown in Figure 5, it deals with fundamentals related to the overall management, software organization, its business, people management, strategic management of

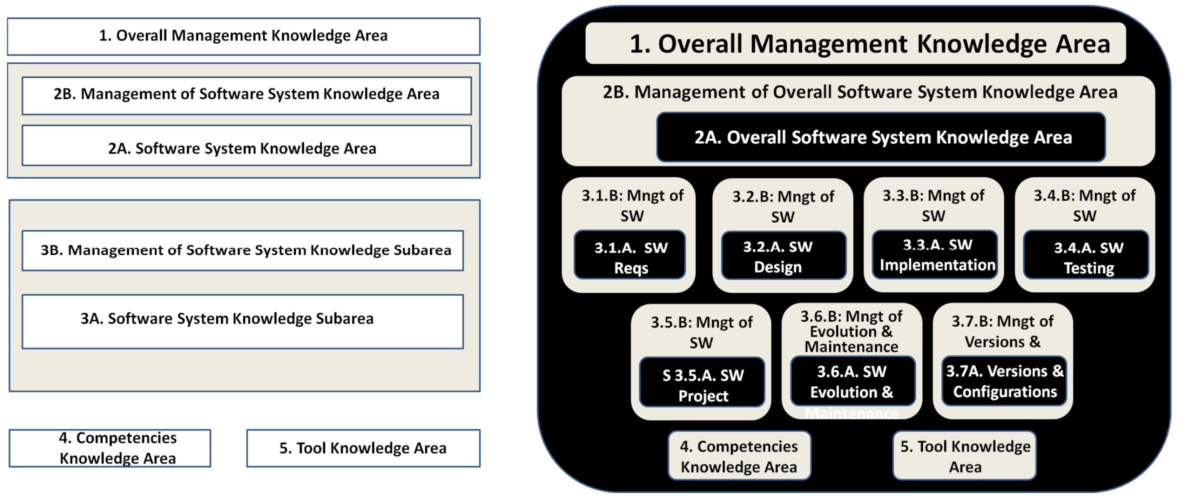

1. Overall Management Knowledge Area
2B. Management of Software System Knowledge Area
2A. Software System Knowledge Area
3B. Management of Software System Knowledge Subarea
3A. Software System Knowledge Subarea
4. Competencies Knowledge Area
5. Tool Knowledge Area
1. Overall Management Knowledge Area
2B. Management of Overall Software System Knowledge Area
2A. Overall Software System Knowledge Area
3.1.B: Mngt of SW
3.1.A. SW Reqs
3.2.B: Mngt of SW
3.2.A. SW Design
3.3.B: Mngt of SW
3.3.A. SW Implementation
3.4.B: Mngt of SW
3.4.A. SW Testing
3.5.B: Mngt of SW
S 3.5.A. SW Project
3.6.B: Mngt of Evolution & Maintenance
3.6.A. SW Evolution & Maintenance
3.7.B: Mngt of Versions &
3.7A. Versions & Configurations
4. Competencies Knowledge Area
5. Tool Knowledge Area

Figure 3. Our suggestion for structuring software engineering body of knowledge

software systems, and the management of organizational standards and technologies.

2. *Overall Software System*: This area includes BoKs that are necessary for managing the software system as a whole during its entire lifecycle.
3. *Software System Knowledge Subareas*: Here, we include all the knowledge subareas that are necessary for managing various parts of software system within various stages and/or processes. In our model, the knowledge subareas are (1) *Requirements*, (2) *Design*, (3) *Implementation*, (4) *Testing*, (5) *Project*, (6) *Evolution and Maintenance*, and (7) *Versions and Configurations*.
4. *Competencies Knowledge Areas*: This area identifies all the competencies and skills that a software professional should possess. As shown in Figure 4, these include personal abilities, software engineering related competencies, business related competencies and technology related competencies.
5. *Tool Knowledge Area*: This area includes knowledge about tools to be used throughout the software lifecycle.

By studying Figure 4, the astute reader might have noticed that *Overall Software System Area* and *Software System Subareas* follow a uniform underlying structure. Before explaining it, however, we would like to come back to Figure 3 and kindly ask the reader to study knowledge areas 2A – 2B and 3.1.A/3.1.B – 3.7.A/3.7.B. As can be seen there, the two knowledge areas are divided into two parts: (1) part dealing with knowledge about a software system or knowledge about a particular knowledge subarea and (2) part dealing with the knowledge about how to manage the knowledge area or subarea.

When defining the structure for *Overall Software System Area* and *Software System Subareas*, we follow the rules from object-orientation where each object encapsulates data describing the object properties and defines methods required for managing the object's data. In this way, we distinguish between the knowledge about a specific area from the knowledge about how to manage the specific area. We believe that it is very important to distinguish between those two pieces of knowledge. This may be motivated with the *Requirements Knowledge Subarea*. You may know what requirement are and how they are structured, however, you may not know how to manage requirements during various lifecycle phases.

The overall structure of *Overall Software System Area* and *Software System Subareas* is presented in Subfigures 2A, 2B, 3A and 3B in Figure 4 and exemplified with the *Requirements* subarea in Figure 5. The astute reader might have noticed that they have a common underlying structure. Below, we are going to describe it. To facilitate the understanding, we kindly ask the reader to follow the structure in Figure 4 and its exemplification on *Requirements Knowledge Subarea* in Figure 5.

Regarding the knowledge about software system area and software system subareas, the structure includes:

- *Fundamentals* comprising definition, things to know and to do within the knowledge area or subarea and descriptions of the area's properties. The description, in turn, consists of two parts: (1) descriptions of general properties and (2) descriptions of the quality of a specific knowledge area. Examples of fundamentals for *Requirements Subarea* are presented on the left handside of Figure 5.
- *Types of Knowledge Area/Subarea* listing the types of knowledge area. In case of requirements in Figure 5, we have functional, non-functional, emergent requirements and the like.
- *Levels of Knowledge Area/Subarea* identifying levels that are specific for a particular area. In case of requirements, we have user requirements, system requirements, software requirements, requirements items and the like.
- *Documentation* of the Knowledge Area/Subarea listing what needs to be documented, identifying documentation formality, audience and providing documentation templates.

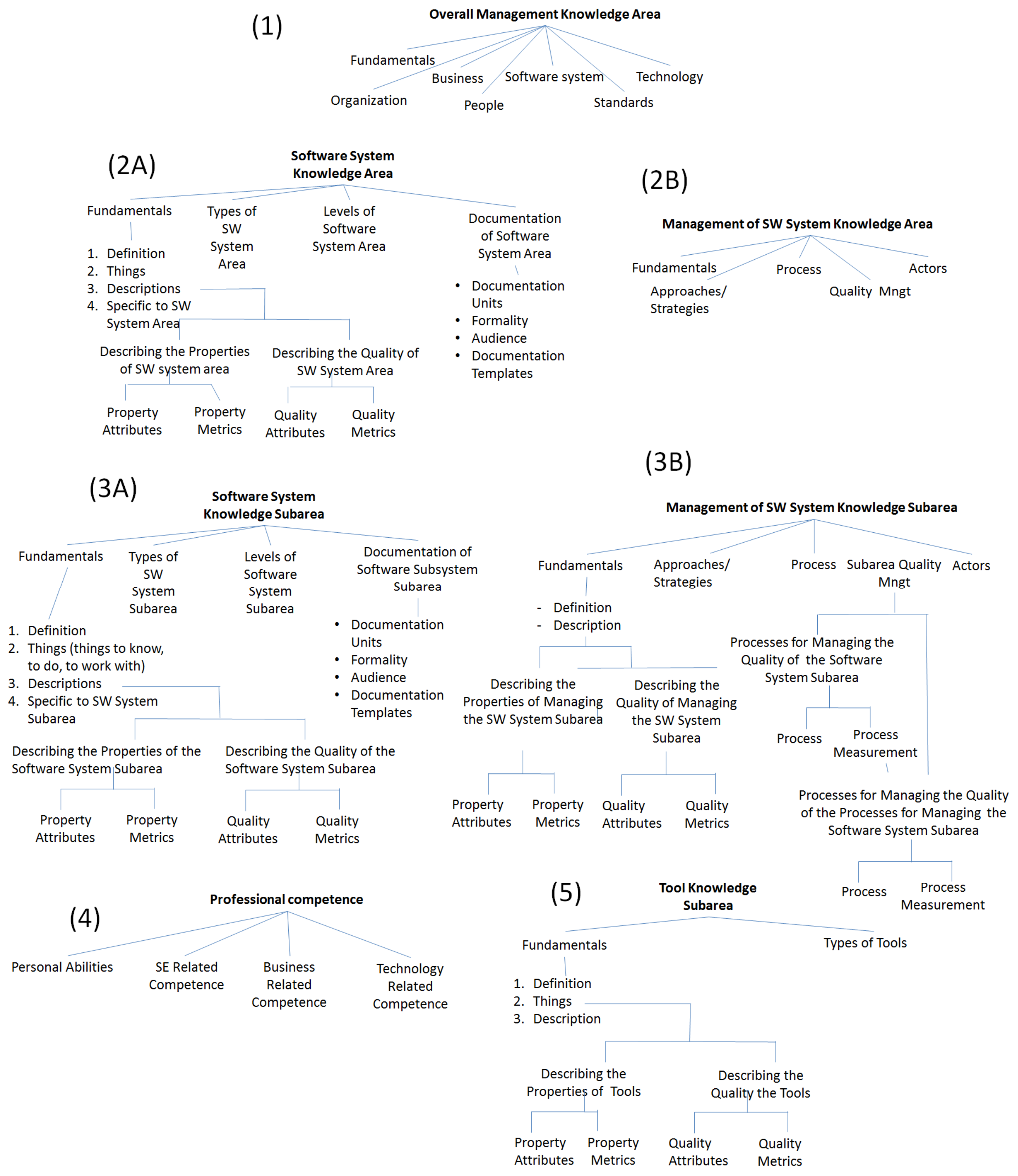
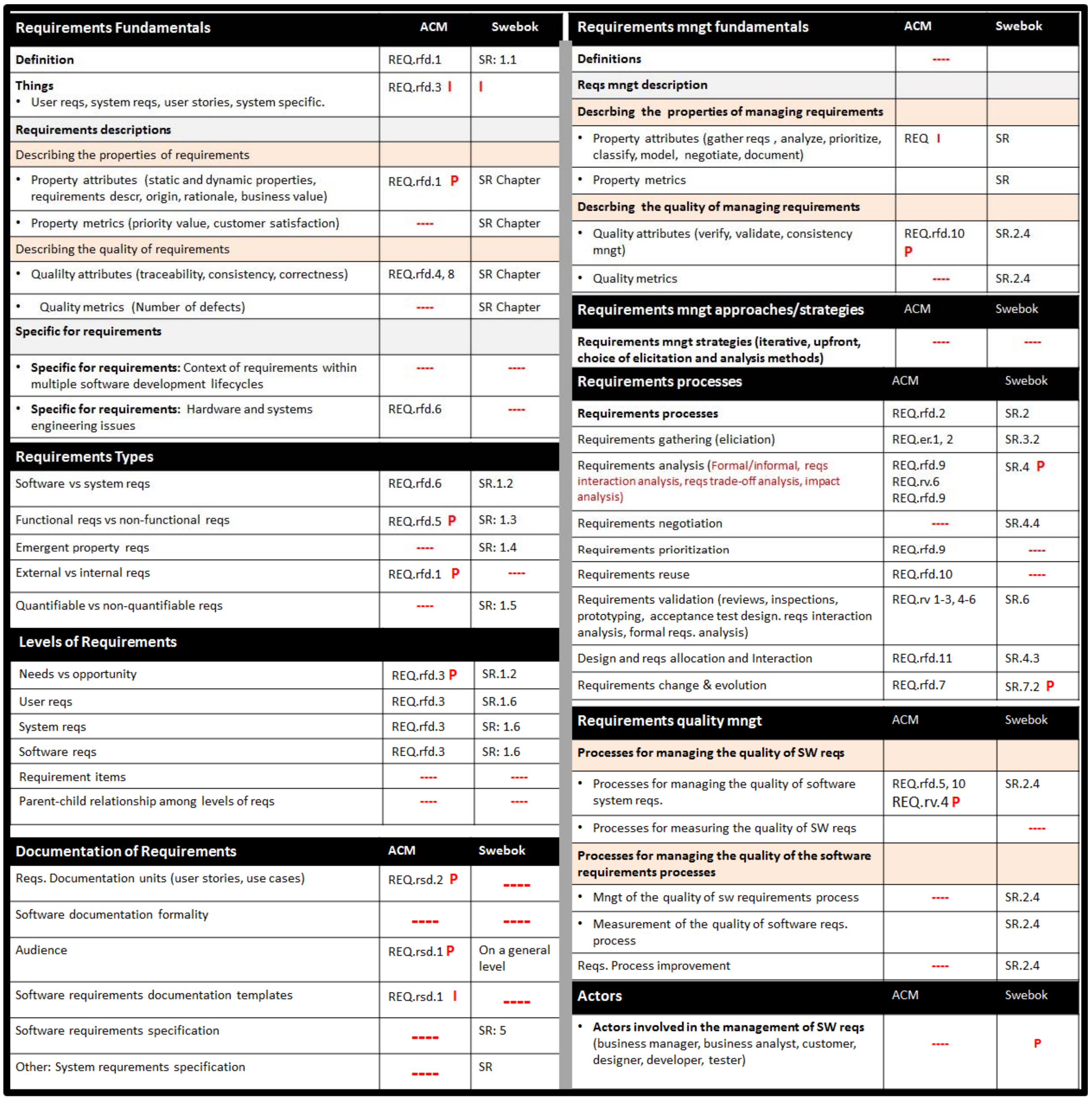

(1)
Overall Management Knowledge Area
Fundamentals
Business
Software system
Technology
Organization
People
Standards
(2A)
Software System Knowledge Area
Fundamentals
1. Definition
2. Things
3. Descriptions
4. Specific to SW System Area
Types of SW System Area
Levels of Software System Area
Documentation of Software System Area
• Documentation Units
• Formality
• Audience
• Documentation Templates
Describing the Properties of SW system area
Describing the Quality of SW System Area
Property Attributes
Property Metrics
Quality Attributes
Quality Metrics
(2B)
Management of SW System Knowledge Area
Fundamentals
Process
Actors
Approaches/ Strategies
Quality Mngt
(3A)
Software System Knowledge Subarea
Fundamentals
Types of SW System Subarea
Levels of Software System Subarea
Documentation of Software Subsystem Subarea
1. Definition
2. Things (things to know, to do, to work with)
3. Descriptions
4. Specific to SW System Subarea
• Documentation Units
• Formality
• Audience
• Documentation Templates
Describing the Properties of the Software System Subarea
Describing the Quality of the Software System Subarea
Property Attributes
Property Metrics
Quality Attributes
Quality Metrics
(3B)
Management of SW System Knowledge Subarea
Fundamentals
Approaches/ Strategies
Process
Subarea Quality Mngt
Actors
- Definition
- Description
Describing the Properties of Managing the SW System Subarea
Describing the Quality of Managing the SW System Subarea
Property Attributes
Property Metrics
Quality Attributes
Quality Metrics
Processes for Managing the Quality of the Software System Subarea
Process
Process Measurement
Processes for Managing the Quality of the Processes for Managing the Software System Subarea
Process
Process Measurement
(4)
Professional competence
Personal Abilities
SE Related Competence
Business Related Competence
Technology Related Competence
(5)
Tool Knowledge Subarea
Fundamentals
Types of Tools
1. Definition
2. Things
3. Description
Describing the Properties of Tools
Describing the Quality the Tools
Property Attributes
Property Metrics
Quality Attributes
Quality Metrics

Figure 4. Our suggestion for structuring software engineering knowledge areas



Requirements Fundamentals	**ACM**	**Swebok**
Definition	REQ.rfd.1	SR: 1.1
Things • User reqs, system reqs, user stories, system specific.	REQ.rfd.3 I	I
Requirements descriptions		
Describing the properties of requirements		
• Property attributes (static and dynamic properties, requirements descr, origin, rationale, business value)	REQ.rfd.1 P	SR Chapter
• Property metrics (priority value, customer satisfaction)	----	SR Chapter
Describing the quality of requirements		
• Qualilty attributes (traceability, consistency, correctness)	REQ.rfd.4, 8	SR Chapter
• Quality metrics (Number of defects)	----	SR Chapter
Specific for requirements		
• **Specific for requirements:** Context of requirements within multiple software development lifecycles	----	----
• **Specific for requirements:** Hardware and systems engineering issues	REQ.rfd.6	----
Requirements Types		
Software vs system reqs	REQ.rfd.6	SR.1.2
Functional reqs vs non-functional reqs	REQ.rfd.5 P	SR: 1.3
Emergent property reqs	----	SR: 1.4
External vs internal reqs	REQ.rfd.1 P	----
Quantifiable vs non-quantifiable reqs	----	SR: 1.5
Levels of Requirements		
Needs vs opportunity	REQ.rfd.3 P	SR.1.2
User reqs	REQ.rfd.3	SR.1.6
System reqs	REQ.rfd.3	SR: 1.6
Software reqs	REQ.rfd.3	SR: 1.6
Requirement items	----	----
Parent-child relationship among levels of reqs	----	----
Documentation of Requirements	**ACM**	**Swebok**
Reqs. Documentation units (user stories, use cases)	REQ.rsd.2 P	----
Software documentation formality	----	----
Audience	REQ.rsd.1 P	On a general level
Software requirements documentation templates	REQ.rsd.1 I	----
Software requirements specification	----	SR: 5
Other: System requrements specification	----	SR

Requirements mngt fundamentals	**ACM**	**Swebok**
Definitions	----	
Reqs mngt description		
Descrbing the properties of managing requirements		
• Property attributes (gather reqs , analyze, prioritize, classify, model, negotiate, document)	REQ I	SR
• Property metrics		SR
Descrbing the quality of managing requirements		
• Quality attributes (verify, validate, consistency mngt)	REQ.rfd.10 P	SR.2.4
• Quality metrics	----	SR.2.4
Requirements mngt approaches/strategies	**ACM**	**Swebok**
Requirements mngt strategies (iterative, upfront, choice of elicitation and analysis methods)	----	----
Requirements processes	**ACM**	**Swebok**
Requirements processes	REQ.rfd.2	SR.2
Requirements gathering (eliciation)	REQ.er.1, 2	SR.3.2
Requirements analysis (Formal/informal, reqs interaction analysis, reqs trade-off analysis, impact analysis)	REQ.rfd.9 REQ.rv.6 REQ.rfd.9	SR.4 P
Requirements negotiation	----	SR.4.4
Requirements prioritization	REQ.rfd.9	----
Requirements reuse	REQ.rfd.10	----
Requirements validation (reviews, inspections, prototyping, acceptance test design. reqs interaction analysis, formal reqs. analysis)	REQ.rv 1-3, 4-6	SR.6
Design and reqs allocation and Interaction	REQ.rfd.11	SR.4.3
Requirements change & evolution	REQ.rfd.7	SR.7.2 P
Requirements quality mngt	**ACM**	**Swebok**
Processes for managing the quality of SW reqs		
• Processes for managing the quality of software system reqs.	REQ.rfd.5, 10 REQ.rv.4 P	SR.2.4
• Processes for measuring the quality of SW reqs		----
Processes for managing the quality of the software requirements processes		
• Mngt of the quality of sw requirements process	----	SR.2.4
• Measurement of the quality of software reqs. process		SR.2.4
Reqs. Process improvement	----	SR.2.4
Actors	**ACM**	**Swebok**
• **Actors involved in the management of SW reqs** (business manager, business analyst, customer, designer, developer, tester)	----	P

Figure 5. Exemplifying *Requirements Knowledge Subarea* using our model and mapping it onto SWEBOK and ACM/IEEE Curriculum. P stands for partially, I stands for implicitly and minus stands for absence

Regarding the knowledge about the management of software system area and software system subareas, the structure includes:

- *Fundamentals* comprising definition, and descriptions of the management of the area/subarea. The description, in turn, consists of two parts: (1) descriptions of the general management properties and (2) descriptions of the quality of managing a specific knowledge area. Examples of fundamentals are presented on the right handside of Figure 5.
- *Approaches/Strategies* for managing the knowledge area/subarea. As shown in Figure 5, requirements may be managed upfront or iteratively.
- *Process* listing all the processes that are relevant for managing the knowledge area. In case of requirements, we have processes like requirements gathering, analysis, negotiation, prioritization, reuse, and the like.
- *Process quality management* identifying the processes for managing the quality of the area/subarea. On purpose, we have distinguished quality management

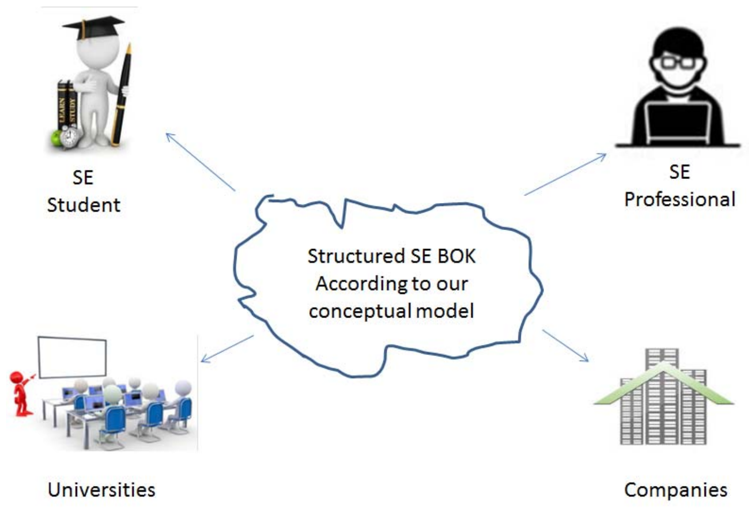


Figure 6. Illustrating usage of our model

processes as a separate BoK from the processes for managing the specific area/subarea. We motivate this with the fact that the management of the quality of the requirements gathering process is not the same as the requirements gathering process. Here, we identify two groups of quality management processes (1) management of the quality of the software system area/subarea and (2) management of the quality of the management of the software system area/subarea. As shown in Figure 5, the first group focuses on evaluating and measuring the quality of requirements whereas the second group focuses on evaluating and measuring the quality of the processes used for managing requirements. These are two separate things. You may have a high quality process but you may have a low quality software system or vice versa.

- *Actors* identifying all the roles that take part within a particular area or subarea. In case of requirements, we have business managers, business analysts, customers and the like.

Regarding the structure of the remaining knowledge areas, they are for now slightly different. However, they follow the common template as much as possible. Below, we motivate their underlying structures.

- The *Competencies* knowledge area lists the knowledge and skills that students must possess. It does not include the management of students' competencies. If we included it, it would rather deal with how universities develop and manage the students' competencies. From the curriculum perspective, it is not relevant. The management of people as software professionals, however, is included in the *Overall Management Knowledge Area* (see (1) in Figure 3 and 4).
- The *Tools* knowledge area does not include the management part, not yet. We believe that the educational programmes will have difficulties in teaching how to manage tools. We are however open for changes in the future.
- *Overall Management Knowledge Area* only lists the strategic elements required for managing organizations, businesses, people and the like. This knowledge is required, however, on a very general level. In reality, other non-software professional roles will possess this knowledge. For software engineering graduate students, it is enough that they are acquainted with the very basics. We are however open for evolving this part in the future, if protests from the software engineering community arise.

IV. Our Model vs SWEBOK and ACM/IEEE Curriculum

Our conceptual model of the software engineering body of knowledge has been mapped onto the BoKs of SWEBOK and ACM/IEEE Curriculum. The model and its mapping has resulted in almost 100 page document. Due to space restrictions, we cannot present it herein. However, we have succeeded to present part of our model, the part dealing with *Requirements Knowledge Subarea* which we then have mapped onto the two standards.

As can be seen in Figure 5, many of the items within the *Requirements Knowledge Subarea* have not been implemented by SWEBOK and ACM/IEEE Curriculum. There are also areas that are totally missing in the two standards. These are *Overall Management Knowledge Area* (Area 1 in Figure 3) and *Software System Area* (Area 2A and 2B in Figure 3). It seems that both standards are more focused on the knowledge subareas and have not put enough effort on the overall management of the software organizations, their businesses and people and the holistic management of a software system.

Right now, we are in the process of further detailing our model. Our goal is to achieve a basis for a one-to-one mapping of the BoKs onto educational programmes and curricula. For instance, *Requirements Analysis* listed on the right handside of Figure 5 needs to be more granular. Each BoK such as formal/informal requirements analysis, requirements interaction analysis, requirements trade-off analysis, impact analysis requires a separate field and attention in its own right. Only in this way, we may see which part of the requirements analysis has been offered by an educational programme.

Even if the model is going to be further detailed in the future, we have found it very easy to use when evaluating the TIDAB programme. It took us much time to develop the model, however, when finally being ready with the model, it took us almost no time at all to evaluate our educational programme.

V. Final Remarks

In this paper, we have pointed out some of the problems when using SWEBOK and ACM/IEEE Curriculum while evaluating KTH TIDAB undergraduate programme [1][7][9]. The problems deal with lack of an underlying common structure in the two standards and difficulties in determining whether particular BoKs have been fulfilled. This has forced us to create our own model that will provide a basis for defining a curriculum for the TIDAB programme [9].

We believe that our model is useful in many contexts. By enabling a one-to-one mapping of the BoKs onto the curricula, it may help many actors. As illustrated in Figure 6, our model is useful for students for finding out how much they are progressing during their studies and for mapping out their knowledge after having graduated. It is useful for software professionals for finding out how much their competencies and skills progress during their lifelong profession.

Universities would use the structured body of knowledge for defining their curricula and for specifying which BoKs and in what depth of knowledge are important for particular undergraduate programmes. Graduate diplomas could include attachments of all the BoKs that students have learned during their studies. This would facilitate the employment process.

Companies often complain that graduate students possess too little knowledge within software engineering. When employing graduates, they have difficulties in finding out the level and depth of the applicants' software engineering competence [5][6]. A document specifying in detail depth the knowledge of the applicants would substantially help them in finding and employing individuals with the right competencies and skills.

Right now, our model is still under development and it needs be validated. We however strongly believe in it and we claim that it is possible to find a uniform underlying structure for most of the software engineering bodies of knowledge. For this reason, as a next step, we invite the software community to join us, further improve the model and complement it with a system for grading the depth of software engineering knowledge. Only in this way, we will have insight into what a specific individual knows or does not know and how s-he should evolve his/her knowledge. We will also be able to compare various programmes on an international level and employers will be better informed about whom they employ.

References

[1] ACM/IEEE, "Software Engineering 2014, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering", 2014, https://www.acm.org/education/se2014.pdf, retrieved on Jan 28, 2017.

[2] T. Hilburn and D Bagert, "A Software Engineering Curriculum Model", Proc. Frontiers in Education Conference (FIE 99), IEEE, Nov. 1999, vol. 1, pp. 12A4/6-12A411, doi: 10.1109/FIE.1999.839269.

[3] G. Hislop, et.al., "Work in Progress - Problems and Progress in Software Engineering Curriculum Materials", Proc. Frontiers in Education Conference, 2004, vol. 2, pp. F2C-F23, doi: 10.1109/FIE.2004.1408592.

[4] IEEE, SWEBOK V3.0, Guide to the Software Engineering Body of Knowledge,http://www4.ncsu.edu/~tjmenzie/cs510/pdf/SWEBOKv3.pdf, retrieved on Jan. 26, 2017.

[5] S. Koolmanojwong, B. Boehm, "Educating Software Engineers to Become System Engineers", Proc. IEEE-CS Conference on Software Engineering Education and Training (CSEE&T'11), 2011, pp. 209-2018, doi: 10.1109/CSEET.2011.5876089.

[6] R Leblanc, A. Sabel, T Hilburn, and A McGerrriek, "IEEE-CS/ACM computing curricula - software engineering volume", Proc. Frontiers in Education Conference, Dec. 2003, vol. 3, pp. S3C_24-S3C_27, doi: 10.1109/FIE.2003.1265988.

[7] Liem, I, et.al., "Reshaping Software Engineering Education towards 2020 Engineers", Proc. IEEE 27th Conference on Software Engineering Education and Training (CSEE&T14), 2014, pp. 171-174, doi: 10.1109/CSEET.2014.6816797.

[8] K. Robinson, P. Ho, "Software Engineering or Soft Engineering?", Proc. IEEE-CS Conference on Software Engineering Education and Training (CSEET&T'11), 2011, pp. 459-466, doi: 10.1109/CSEET.2011.5876125.

[9] TIDAB, Degree Programme in Computer Engineering, 2014, https://www.kth.se/social/program/tidab/, retrieved on Jan. 28, 2017.