SWEBOS – The Software Engineering Body of Skills

http://dx.doi.org/10.3991/ijep.v5i1.4047

Yvonne Sedelmaier and Dieter Landes
University of Applied Sciences and Arts, Coburg, Germany

***Abstract*—The development of complex software systems requires a mixture of various technical and non-technical competencies. While some guidelines exist which technical knowledge is required to make a good software engineer, there is a lack of insight as to which non-technical or soft skills are necessary to master complex software projects. This paper proposes a body of skills (SWEBOS) for software engineering. The collection of necessary skills is developed on the basis of a clear, data-driven research design. The resulting required soft skills for software engineering are described precisely and semantically rich in a three-level structure. This approach guarantees that skills are not just characterized in a broad and general manner, but rather they are specifically adapted to the domain of software engineering.**

***Index Terms*—Software Engineering Education; Soft Skills in Software Engineering; Non-Technical Skills; Description of Competencies; SWEBOK**

I. INTRODUCTION

Software is a core ingredient of nearly any part of our everyday life. Software systems, however, need to be developed by highly skilled individuals. Consequently, education in software engineering in order to acquire and exercise the required skills plays an important role in university education. Traditionally, universities laid their main emphasis in software engineering education on technical expertise, such as programming or testing. In recent years, however, it became increasingly evident that non-technical, also known as soft, skills are equally important as software is developed in teams of individuals who need to interact with each other and with various stakeholders such as, e.g., customers or users of their software.

So far, however, there is no subject didactics for software engineering which would allow for a more systematic choice of didactical approaches in software engineering education [1]. Such a subject didactics would encompass competency profiles which constitute the targets for software engineering education, in conjunction with didactical approaches that are likely to support the achievement of these goals. Our approach towards the development of such a subject didactics encompasses three main areas of research, namely the identification and description of relevant competencies for software engineering, experiments with didactical approaches that presumably address these competencies, and an assessment framework to evaluate didactical approaches with respect to their eligibility, given particular target competencies.

In the following, we will concentrate on the first issue, i.e. a competency profile for software engineering. While there are some guidelines as to which technical expertise is required for a software engineer, e.g. in the Software Engineering Body of Knowledge (SWEBOK) [2, 3], the non-technical side of skills is less well understood.

This contribution presents a framework to describe software engineering competencies that spans various degrees of abstraction. In order to fill this framework with contents, we rely on a data-driven approach which is based on Grounded Theory [4]. Finally, we outline preliminary results with respect to a "Software Engineering Body of Skills", i.e. a prioritized account and characterization of non-technical skills that are specifically instantiated with respect to the domain of software engineering. As a consequence of our findings, we argue that it is reasonable to distinguish generic non-technical skills, such as presentation skills, from context-sensitive non-technical skills that exhibit a special flavor in software engineering and in conjunction with specific technical skills.

II. DEFINITION OF TERMS

Many different definitions and descriptions concerning competency, soft skills, knowledge, expertise etc. exist in pedagogy. In particular, "competency" is one of the most popular terms since the beginning of the 21st century and the introduction of the Bologna process. Yet, it is also one of the most confusing terms used in several disciplines and also by lay people. Therefore, we first explain our understanding of several terms used in this paper and their interrelationships.

Competency denotes a comprehensive capability to act appropriately in complex situations. The capability to act includes technical knowledge, also called factual knowledge. The capability to cope with complex and new situations also presupposes additional skills, which are often subdivided into social, personal, and methodological competence [5–7].

In this paper, the term "competency" is not used when factual or technical knowledge is described. Likewise, according to Weinert [8] (p. 35) "skill is an ability to perform complex motor and/or cognitive acts with ease, precision, and adaptability to changing conditions". Following this view, neither soft skills, nor factual knowledge in isolation are competencies. Competencies can only come into existence when both interact: "Competency" presupposes technical or factual knowledge and also soft skills. Moreover, competency encompasses the context, emotional elements, and also possesses an ethical, normative component. Competency enables individuals to analyze complex and new situations, to find creative potential solutions, and to decide on one way of action, in due consideration of causes and consequences. Competency also includes the willingness and motivation to act autono-

mously and based on self-initiative after a cognitive analysis of a situation.

We distinguish context-sensitive soft skills, generic soft skills, and factual knowledge. Generic soft skills are abilities that are largely independent of software development and are relevant for other disciplines, too. Presentation skills are a typical example: they are equally relevant for a social worker, a businessperson, or a software engineer. Presentations follow the same rules regardless of the context in which they are given. For instance, a presenter should speak to his audience and show legible slides, irrespective of whether a software architecture or a new washing machine is presented. In contrast, domain-independence does not hold for context-sensitive soft skills. Skills in this category exhibit a special, unique profile in the context of software engineering. A lack of these skills leads to specific consequences in the software engineering process. Even if a software architecture is not presented properly, colleagues will still understand what to do. Yet, if communication skills or the ability to solve problems or conflicts are available only to an insufficient degree, the whole software engineering project may fail. For instance, dealing with requirements for a software engineering project requires technical knowledge such as process modelling notations or UML use case diagrams. Yet, if software engineers were not capable of eliciting requirements from customers by applying communication techniques, there would be nothing to model. Therefore, it is important to understand the precise meaning of communication skills in a software engineering context.

Should software engineers be able to talk in three different languages? Or should they be able to write good technical documentation? Or must they recognize and solve misunderstandings arising from badly formulated requirements? And how can they formulate requirements as clearly as possible? Which communication techniques may help to cover these challenges?

The combination of various context-sensitive soft skills yields a specific profile of software engineering competencies. Because context-sensitive soft skills are closely related to technical knowledge, students must be trained and advanced with respect to these skills in the context of software engineering. Only in relation to technical expertise and the software development process will students really become aware of the inherent problems.

Factual knowledge consists of basic and advanced facts and methods from a specific subject domain. For software engineering, factual knowledge is, e.g., about the language features of a particular programming language, key features of a process model such as SCRUM or the waterfall model, or particular requirements elicitation techniques and their characteristics.

SWEBOK [2, 3] provides a catalog of relevant factual knowledge in software engineering and offers a structure to this body of knowledge by organizing knowledge items in various knowledge areas.

III. EXISTING GUIDELINES FOR TEACHING AND LEARNING SOFTWARE ENGINEERING

Traditionally, knowledge areas in SWEBOK span the complete software development lifecycle from requirements engineering to software testing, implementation, and related disciplines. Furthermore, the required level of competencies is sketched on the basis of Bloom's taxonomy [9]. On this basis, lecturers get a first bit of evidence as to whether students are supposed to only remember some piece of information in software engineering, to develop a deep understanding of the topic, or to be able to analyze a concept and disassemble it into relevant parts.

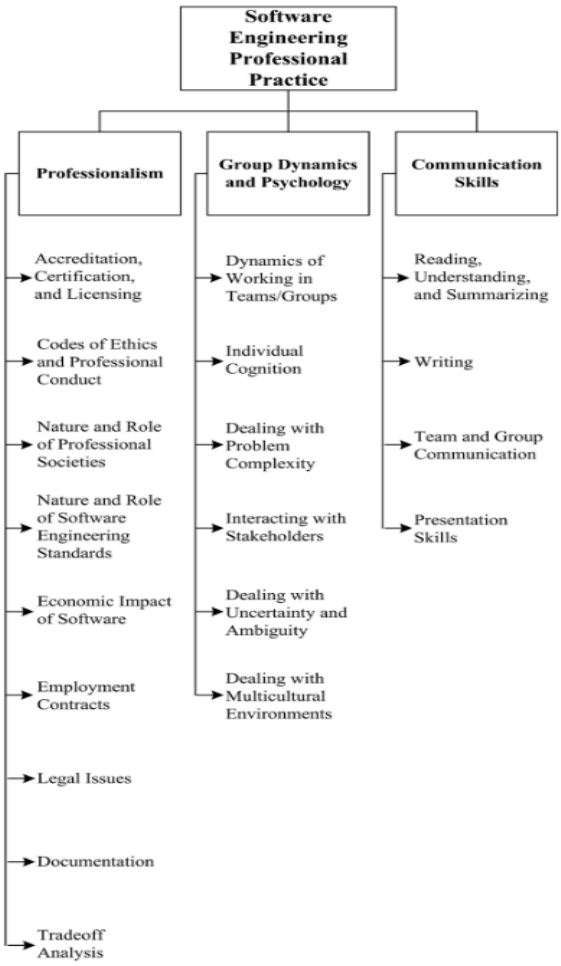

Figure 1. Non-technical skills in SWEBOK v3

The new SWEBOK version (SWEBOK v3) [3] contains a new knowledge area called "Software Engineering Professional Practice" which focusses on "the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner. The study of professional practice includes the subareas of professionalism, group dynamics and psychology, and communication skills".

Taking non-technical skills into account at all is definitively a step into the right direction. Yet, the SWEBOK approach still exhibits several deficiencies.

For one thing, the non-technical skills mentioned in the new SWEBOK version seem to be the result of a qualitative literature study. But it is not described which method was used to arrive at exactly those knowledge areas or skills that are covered in SWEBOK. Such an approach would be completely inappropriate in social sciences. Social sciences presuppose a traceable research design in order to arrive at trustable, valid, and acceptable results. In social sciences it is inevitable to disclose the way in which research results were derived in order to make them plausible. Not only the results count, but the approach taken to get them is even more important. Only a structured, relia-

ble and valid research process can provide reliable data. And only reliable data are valuable data for understanding the field of research. As a core aim of social sciences, data must mirror reality faithfully. Thus, SWEBOK does not meet the requirements of social sciences because it is not traceable where data come from and how they were derived. So there will always be doubt if the findings in SWEBOK depict the world correctly because there is no possibility of tracing and checking how they were developed. In SWEBOK it stays unclear which data were used, where they came from, and how they were merged. Therefore, SWEBOK is only a relatively weak guideline due to its lack of sound scientific underpinning.

SWEBOK tries to "promote a consistent view of software engineering worldwide" [3] (p. xxxi). In doing so, SWEBOK tries to identify the least common denominator of all disciplines that are concerned with software development, i.e. computer scientists, electrical or mechanical engineers and experts from many other domains. As a consequence, specific priorities and biases in some of these disciplines are neglected. SWEBOK describes required knowledge quite general and abstract to suit all domains.

Also, relevant non-technical skills are characterized by a hierarchy of headings, such as "Team and Group Communication" and an additional paragraph of fairly general explanatory prose text (see Fig.1). There is neither any prioritization, nor an indication of how non-technical and technical skills do interact, let alone a scale that would allow the determination of the degree to which a competency is exhibited by an individual. Furthermore, SWEBOK treats skills in a merely descriptive fashion. There is no guideline as to how required competencies, be it technical or non-technical, might be broken down into intended learning outcomes that can be addressed in university education for future software engineers.

Besides, SWEBOK addresses a target audience other than university students, namely software professionals with four years of work experience.

SWEBOK also wants to "provide a foundation for curriculum development" [3] (p. xxxi). Additional recommendations for a software engineering curriculum can be found in "Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering" [10]. Yet, the main criticism that can be expressed against SWEBOK also applies to the IEEE/ACM curriculum.

Both SWEBOK and the IEEE/ACM curriculum can propose contents for a university curriculum in software engineering, but they are not sufficient as they focus on technical expertise on a very abstract level. Worst of all, however, is their lack of attention towards the non-technical skills that are required in software engineering. Both handbooks give only superficial recommendations, if any, of which soft skills a software engineer should have and what a particular soft skill exactly means. In addition, as pointed out above, there is no indication of which methods were used to derive the recommendations in the SWEBOK and the IEEE/ACM curriculum.

Software engineering education at universities is the main focus of the research project EVELIN (Experimental improVEment of Learning software engINeering). For this purpose, the approach taken in SWEBOK seems to be insufficient. Therefore, we propose a different approach to identify and characterize competencies in Software Engineering which will be detailed in the following.

IV. DESCRIPTION OF CONTEXT-SENSITIVE NON-TECHNICAL SKILLS IN SWEBOS

We have chosen a bottom-up research design for structuring and describing context-sensitive non-technical skills in software engineering [11].

The SWEBOS structure has to meet several objectives: First of all, it should support the development of didactical approaches in software engineering education. SWEBOS is intended to lay a basis for a teaching goal- and competency-oriented approach. Thus, SWEBOS should help to describe and understand the targets of software engineering education at universities of applied sciences. This implies that competencies are specified precisely such that they can be measured. SWEBOS follows a data-driven scientific approach in order to be not just a simple smorgasbord of meaningless phrases, but rather foster a deep understanding of which competencies a software engineer must have. SWEBOS should help to understand the semantics of context-sensitive soft skills in a software engineering context. The context of software engineering, however, may vary. For instance, software engineering for embedded systems is different from software engineering for workflow systems. Therefore, the peculiarities of the context may lead to different emphasis on some competencies within SWEBOS. SWEBOS should be a clearly organized tool that nevertheless contains rich descriptions [12, 13] of relevant competencies, depending on the context. Furthermore, it should impose a structure on required competencies in software engineering on several levels of abstraction in order to ensure a uniform description of competencies.

A. Grounded Theory as Research Methodology

SWEBOK's approach of gaining data – presumably by using qualitative content analysis according to Mayring [14] – seems to be inadequate for our research goals. In EVELIN, we want to identify required competencies in software engineering and strive for a better understanding of what abstract terms like team competence might mean in the context of software engineering. To that end, we want to follow a defined and traceable research design in order to arrive at sound results, based on data from real practice.

To achieve these goals, Grounded Theory [4] seems to be an appropriate research methodology as it is data-driven. Grounded Theory is targeted on establishing a new theory by building and testing hypotheses rather than verifying an existing theory. Thus, the main focus lies on understanding the field of research.

The data sample evolves during the research process. Initially, there is no fixed sample but rather an idea who should be asked in the first place about what particular issue. During the research process researchers get new hints and decide on who should be asked next. As a consequence, the sample is never complete before the research ends. Insights that are gained from the data control the ensuing research process. Also research questions may develop during the process by obtaining unexpected information from the field of research. Grounded Theory is based on data and requires adapting the research process to the data.

Consequently, it is also possible to combine several research methods. There is no exclusive emphasis of qualitative or quantitative research methods. The decision on which methods to use is exclusively driven by the consideration which methods are likely to provide the most interesting data for understanding the field of research.

Grounded Theory seems to be an adequate research design for understanding desired competencies in software engineering. Our research aims at understanding required competencies in software engineering and their precise meaning. We arrived at a specific understanding and definition of competencies (see sec. II).

B. Research Process in Detail

We currently pursue a qualitative research design based on Grounded Theory which consists of two main data collection activities (see Fig. 2).

In a first step, we returned to a small data sample of informal conversations which were originally conducted in order to adapt technical knowledge areas from SWEBOK to university education. We analyzed this material again, but now with a focus on non-technical skills. In a bottom-up fashion, we arrived at a first version of a code system and preliminary indications what is really necessary for software engineering in industry. The code system was built by tagging those text sections in the interviews where non-technical skills are explicitly or implicitly described or mentioned. The code system also served as an aid for structuring and clustering identified competencies into a competence profile. Fig. 3 shows the code-system we extracted from our research data by applying this methodology.

In a second step, we conduct guided interviews with practitioners from software companies who are, among other things, in charge of human-resource issues. Now, interviews focus on necessary context-sensitive soft skills. The results of our initial analysis of the material gained in the previous step were compiled into an interview guideline to provide some structure for new interviews. The interview guideline contains questions such as:

- Please describe a perfectly normal working day of yours, including your tasks and activities.
- Where did you learn this? How did you become a good software engineer?

Additional interviews are conducted for two main goals: on the one hand, we intend to include additional points of view with respect to context-sensitive soft skills, and on the other hand, we want to obtain a deeper understanding what these competencies exactly mean. In particular, this research design allows us to analyze detailed data, giving us a deep understanding of the requirements in daily software engineering business.

As a further result, we are able to refine our code-system. The existing code-system was merged with the results of the interviews of the second phase. On this basis, the code-system led to semantically rich, thick descriptions [13] of required competencies in software engineering.

Thus, we extracted a deep understanding which competences a software engineer must have and what is meant by these competencies. Notably, the "definitions" of competencies in SWEBOS are based on research data and reflect the real world. This paves the way to deduce concrete competencies in order to develop curricula and assess competencies.

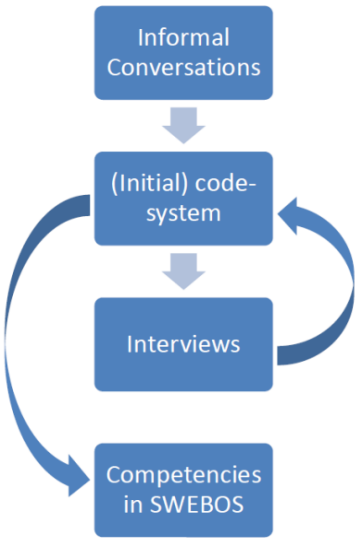

Figure 2. Research Process in Detail

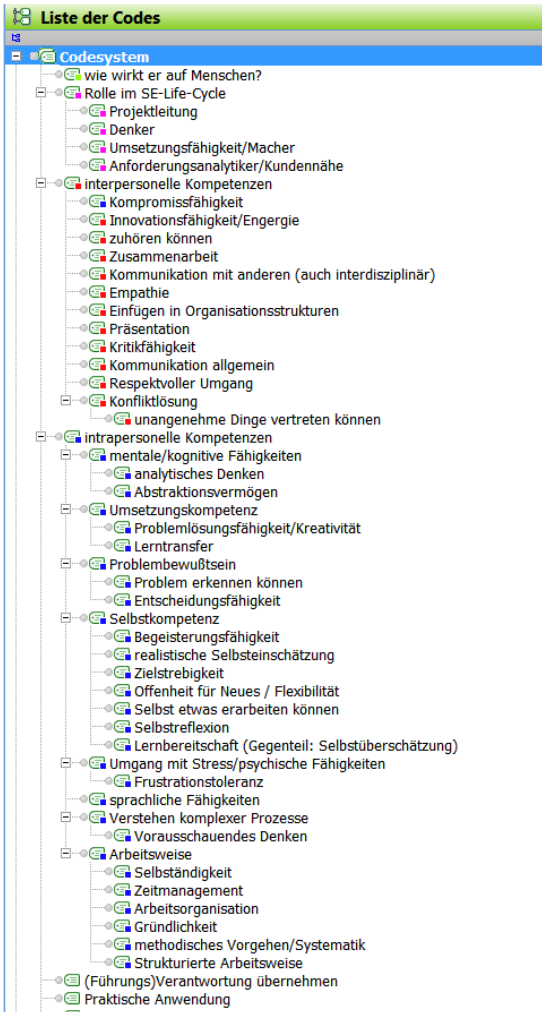

Figure 3. Code system (in German)

Our findings substantiate that it is reasonable to distinguish technical knowledge, context-sensitive soft skills, and generic soft skills (see sec II).

V. REQUIRED COMPETENCIES IN SOFTWARE ENGINEERING

A. Structure of SWEBOS

SWEBOS aims at describing required soft skills in software engineering in a precise fashion, not only by using empty, overly general phrases or a listing of meaningless buzzwords. SWEBOS wants to pinpoint what exactly is expected from a software engineer by providing a semantically rich description [12] of the meaning of relevant soft skills.

One possibility to describe competencies is a textual definition like in glossaries or dictionaries. This approach is insufficient for our purpose because this would result in a collection of somewhat isolated definitions without any interrelations in between. Furthermore, it is difficult to obtain a big picture of which mixture of competencies makes up a competent software engineer.

Interrelationships between particular skills can be highlighted well through a graph-based representation [15]. Unfortunately, the precise definition of individual competencies typically gets lost in a graph since the nodes are generally tagged with a single noun. Only abstract terms can be used in a graph, otherwise things become intransparent and confusing.

Apparently, an adequate structure for describing required competencies must meet two conflicting requirements. One the one hand, the required competencies should be described in a clear and understandable, but not too complex fashion. On the other hand, SWEBOS should give the reader a deep understanding of what is meant when a competent software engineer is described. SWEBOS also requires a description which depicts interrelationships between particular elements of a competent software engineer without tearing the overall picture to pieces.

In order to strike the balance between these two requirements, SWEBOS identifies required soft skills in software engineering in a data-driven approach and describes them on various levels of abstraction in order to depict the underlying data properly. These layers should not be considered in isolation, but as one rich, thick description as it is used in qualitative social sciences [16–18, 13]. Mills [19] explains the matter as follows: "Many qualitative researchers [...] use the term *thick description* to highlight the necessity of paying attention to significant detail in the process of doing research field work. According to the well-known qualitative research methodologist Norman Denzin, the importance of thick description is that it makes *thick interpretation* possible. It is not just quantity of detail that matters but the illumination such detail can afford. Thick description does not mean accumulating voluminous details about everything that happens to the point of trivia. Description must be balanced by analysis, seeking to establish the significance of actions, behaviors, or events for the participants involved."

This approach of describing competencies as a very complex issue allows the description of a deep understanding of software engineering competencies in a detailed manner, in parallel with an account of the complex interrelations of particular aspects of competencies.

Thus, each non-technical skill is subsumed under a specific category, such as communication skills. This level of abstraction is similar to the hierarchy in SWEBOK v3, but tends to be more exhaustive. The second abstraction level characterizes the non-technical skills by a fairly general prose definition as e.g. in glossaries – again somewhat similar to SWEBOK's approach. On the third level of description, however, the SWEBOS approach provides particular indicators or criteria that are amenable to be used for determining if, or to what extent, a particular competency is present or absent. For instance, an indicator for communication skills might be that "a software engineer shall be capable of resolving conflicts constructively". Thus, non-technical skills are characterized by a set of measurable indicators, in a similar fashion as the specification of non-functional requirements should be accomplished, if done properly.

Descriptions of competencies should be understandable across different domains, from pedagogy through to computer science. Therefore, we have chosen a tabular representation for competency descriptions which enforces some common structure, but still leaves room for prose.

B. Contents of SWEBOS

Our research data indicate that the set of competencies described below are required for software engineering:

TABLE I. COLLABORATION

Competencies for professional collaboration (Z)	
These competencies precede other groups of competencies since they constitute a core element of software engineering practise. Only those individuals who are capable of collaborating with other humans are able to solve the problems that are posed to software engineers. These problems cannot be solved without taking the organizational context into account. Software engineers need to collaborate with other humans and to make appropriate contributions to the overall task, irrespective of their particular roles in a specific project.	
Z1	Software engineers are capable of cooperating with others in a team.
Z2	Software engineerings are capable of and willing to communicate with others, even across disciplinary boundaries.
Z3	Software engineers exhibit empathy and are capable of getting acquainted with uncommon circumstances.
Z4	Software engineers are capable of and willing to fit themselves into given structural and process organizations.
Z5	Software engineers are capable of presenting their ideas and issues from their own area of expertise to others.
Z6	Software engineers are capable of a realistic self-estimation of their professional competencies and know their individual strengths and the limits of their professional competencies.
Z7	Software engineers appreciate the professional competencies of others and behave with respect.

TABLE II. COMMUNICATION

Communicative competencies (K)	
For a similar reason, competencies for professional collaboration with others are followed by communicative competencies of software engineers. Communication is a necessary and inevitable means of information exchange whenever humans collaborate. Since each human being has an individual model of the world and filters incoming information according to this model, ambiguities and misunderstandings are inevitable. Communicative competencies are required in order to alleviate this problem. Software engineers need to be capable of accepting and understanding foreign views of the world and of reacting accordingly. They need to be willing to engage in foreign views of the world and they need to be capable of handling foreign views of the world appropriately.	
K1	Software engineers are capable of handling criticism, i.e. they are capable of appropriately advancing their point of view, of contributing objectively to discussions, and of giving and receiving feedback.
K2	Software engineers are capable of resolving conflicts constructively.

TABLE III. STRUCTURE

Competencies for structuring one's own way of working (S)	
These competencies target the fact that software engineers need to structrue themselves and their way of working in a complex environment. Since software development is extremely based on a division of labour, any involved party is dependent on others' contributions in order to come up with a working final result. Therefore, software engineers need to be capable of organising themselves and of structuring their way of working.	
S1	Software engineers are capable of analytic thinking.
S2	Software engineers are capable of setting goals for themselves and of working towards these goals.
S3	Software engineers are capable of motivating themselves to contribute their share, even in complex workflows, in complex team structures, and over an extended period of time.
S4	Software engineers are capable of accepting responsibilities and of solving problems in a self-directed fashion, even without external push.
S5	Software engineers are capable of planning their time realistically, of setting up schedules, and of completing tasks in an organized manner.
S6	Software engineers are working thoroughly and handle their reposibilities carefully.

TABLE IV. PERSONAL COMPETENCIES

Personal competencies (P)	
This group of competencies targets self-relection and the conscious and goal-oriented handling of individual challenges and obstacles.	
P1	Software engineers reflect on themselves and their capabilities and skills regularly and draw conclusions for future assignments from that.
P2	Software engineers are capable of working calmly and efficiently, even under time pressure or occupational stress.
P3	Software engineers are capable of bearing and coping with setbacks appropriately.
P4	Software engineers are aware of the fact that acquired professional knowledge must be combined with experience for being able to develop complex software systems.

TABLE V. CONSCIOUSNESS OF PROBLEMS

Capability to understand complex processes, systems, and relationships (problem awareness) (V)	
These competencies aim at abstracting complex problem settings and concretizing potential solutions. Software engineers get in touch with many other professional disciplines. Thus, they need to be willing to and capable of thinking outside the box, and of understanding and accepting the signification and necessity of allegedly strange procedures and artifacts.	
V1	Software engineers are capable of abstracting and modelling complex situations.
V2	Software engineers recognize which abstract solution pattern might be applied to a specific situation.

TABLE VI. COMPETENCE TO SOLVE PROBLEMS

Capability to apply one's individual knowledge and skills to concrete and novel situations (Solution competency L)	
Acquired professional knowledge is not complete in every respect and needs to be applied and transferred creatively to novel situations. Situations differ largely. Thus, cook book recipes are not applicable. Rather, potential solutions must be developed and analyzed on the basis of building abstractions and drawing analogies. Therefore, software engineers need to be flexible to respond to novel challenges quickly and must develop and update their knowlegde continuously.	
L1	Software engineers are capable of developing creative potential solutions for professional problem settings.
L2	Software engineers are capable of evaluating different approaches, of choosing the most promising approach, and of pursuing the chosen approach carefully.
L3	Software engineers are willing to and capable of becoming acquainted with novel subjects and areas over their complete professional career in a self-directed manner.
L4	Software engineers exhibit openness towards others and towards novel situations. They are willing to and capable of getting involved in unprecendented and unplanned situations and of responding to these situations flexibly and appropriately.

TABLE VII. ADDITIONAL COMPETENCIES

Additional competencies (W)	
W1	Software engineers accept responsibility for others and for joint projects.
W2	Software engineers are capable of researching required information and of adapting and utilizing identified information in order to solve a specific problem.
W3	Software engineers are capable of gauging the consequences of their activities and of behaving according to social and ethical norms.
W4	Software engineers are capable of expressing themselves appropriately in writing.

VI. SUMMARY AND OUTLOOK

Developing large software systems is a complex undertaking which requires highly skilled individuals to be accomplished. Consequently, education in software engineering in order to acquire and exercise these skills plays an important role in university education. As it turns out, non-technical, or soft, skills are equally important as factual or technical knowledge since software is usually developed in teams of individuals. These individuals need to interact with each other and various stakeholders such as, e.g., customers or users of their software.

Unfortunately, there are no sound guidelines that indicate which non-technical skills are particularly relevant for software engineers. Furthermore, existing recommendations contain only fairly general and very abstract descriptions of the respective skills. In order to address these shortcomings, our research aims at developing a software engineering body of skills (SWEBOS) that characterizes these skills in a precise and semantically rich manner.

The research method for the development of SWEBOS is a data-driven approach based on Grounded Theory [4]. In particular, a series of interviews with practitioners was conducted to provide the required data base. These data were analyzed qualitatively on the basis of a code system, which was in turn developed on the basis of an initial sample of the data. We were able to identify a preliminary list of soft skills that are relevant for software engineers. In particular, our research showed that the three top soft skills in software engineering are:

- Comprehension of the complexity of software engineering processes and understanding of cause-effect relationships;
- Problem-awareness and the capability to develop creative solutions;
- Team competence including communication skills.

We intend to conduct additional interviews to continuously refine and update SWEBOS in the spirit of Grounded Theory. These interviews are also expected to give us an even better understanding of biases towards required competencies in different domains such as mechanical engineering. Understanding such biases helps us to develop and evaluate didactical approaches which are tailormade to the intended learning outcomes of these disciplines.

A major advantage of the SWEBOS approach lies in the fact the results are firmly grounded on the current practice of software engineering.

SWEBOS serves as a basis for competency oriented teaching and learning. SWEBOS will be used to deduce intended learning outcomes in university education. In-

tended learning outcomes are the prerequisite for evaluating and developing adequate didactical approaches to learn and teach software engineering. SWEBOS has already been used to clarify, revise, and extend the intended learning outcomes in a requirements engineering course, resulting in considerable changes of the employed didactical approaches [20, 21].

SWEBOS also establishes the basis for a measurement framework for competencies. We are currently working on SECAT, a software engineering competency assessment tool. SECAT is primarily intended to support the identification of potential improvements in software engineering education [22]. To that end, it tries to determine if and to what extent relevant competencies are present among the participants of a software engineering course. A mismatch between the observed level of competence and the intended learning outcomes indicates that the employed didactical methods might be inappropriate to exercise relevant competencies properly and should be adapted and enhanced.

ACKNOWLEDGMENT

We thank all our interview partners who provided input to SWEBOS, in particular Bernd Hindel (Methodpark, Germany), Florian Höpfl (Softgate, Germany), and Thomas Kammerer (Astrum IT, Germany).

The research project EVELIN is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant no. 01PL12022A.

REFERENCES

[1] P. Figas, S. Knörl, S. Mörtlbauer, Y. Sedelmaier, and I. Schroll-Decker, "Developing a Software Engineering Education As a Didactical Discipline," in 1st European Conference on Software Engineering Education (ECSEE), 2014, to appear.

[2] A. Abran and J. W. Moore, *Guide to the software engineering body of knowledge*. Los Alamitos, Calif: IEEE Computer Society, 2004.

[3] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge Version 3.0 - SWEBOK.* Available: http://www.swebok.org/ (15. July 2014).

[4] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine Transaction, 2009.

[5] H. Orth, *Schlüsselqualifikationen an deutschen Hochschulen:* Konzepte, Standpunkte und Perspektiven. Bielefeld: UVW, Webler, 1999.

[6] L. Reetz, "Zum Zusammenhang von Schlüsselqualifikationen - Kompetenzen - Bildung," *Aus Politik und Zeitgeschichte. Beilage zur Wochenzeitung Das Parlament*, no. 37, pp. 13–20, http://www.sowi-online.de/reader/berufsorientierung/reetz_lothar_1999_zum_zusammenhang_von_schluesselqualifikationen_kompetenzen_bildung.html, 1999.

[7] D. Schneckenberg, *Educating Tomorrow's Knowledge Workers: The Concept of ECompetence and Its Application in International Higher Education*: Eburon, 2008.

[8] F. E. Weinert, *Concepts of Competence:* Definition and Selection of Competencies. Theoretical and Conceptual Foundations (DeSeCo)

[9] B. S. Bloom, M. Engelhart, E. Furst, W. Hill, and D. R. Krathwohl, *Taxonomy of Educational Objectives – The Classification of Educational Goals – Handbook 1: Cognitive Domain.* London, WI: Longmans, Green & Co. Ltd, 1956.

[10] IEEE Computer Society and Association for Computing Machinery ACM, *Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering:* A Volume of the Computing Curricula Series. Available: http://sites.computer.org/ccse/SE2004Volume.pdf(2013, Sep. 24).

[11] Y. Sedelmaier and D. Landes, "A Research Agenda for Identifying and Developing Required Competencies in Software Engineering," *International Journal of Engineering Pedagogy (iJEP)*, vol. 3, no. 2, pp. 30–35, 2013.

[12] S. B. Merriam, *Case study research in education:* A qualitative approach, 1st ed. San Francisco: Jossey-Bass, 1988.

[13] S. B. Merriam, *Qualitative research in practice:* Examples for discussion and analysis, 1st ed. San Francisco: Jossey-Bass, 2002.

[14] P. Mayring, *Qualitative Content Analysis.* Available: http://www.qualitative-research.net/index.php/fqs/article/view/1089/2385.

[15] M. Koch and D. Landes, "Notations for Modeling Educational Goal Profiles," in 1st European Conference on Software Engineering Education (ECSEE), 2014, to appear.

[16] C. Geertz, *The interpretation of cultures:* Selected essays. New York: Basic Books, 1973.

[17] I. Holloway, *Basic concepts for qualitative research*. London, Malden, MA, USA: Blackwell Science, 1997.

[18] Y. S. Lincoln and E. G. Guba, *Naturalistic inquiry*. Beverly Hills, Calif: Sage Publications, 1985.

[19] A. J. Mills, *Encyclopedia of Case Study Research*. Thousand Oaks: Sage Publications, 2010.

[20] Y. Sedelmaier and D. Landes, "A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering," in Requirements Engineering Education and Training, 2014, to appear.

[21] Y. Sedelmaier and D. Landes, "Using Business Process Models to Foster Competencies in Requirements Engineering," in 27th International Conference on Software Engineering Education and Training (CSEE&T), 2014, pp. 13–22.

[22] Y. Sedelmaier and D. Landes, "A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies: SECAT - A Software Engineering Competency Assessment Tool," in 44th Frontiers in Education (FIE), 2014, to appear.

AUTHORS

Y. Sedelmaier holds a diploma in pedagogy with a major focus on adult learning and continuing education at the University of Bamberg, Germany. After ten years working experience in the educational sector and in quality management she is now academic researcher in the project "Experimental improvement of learning software engineering" (EVELIN) and investigates students and their learning processes. Her research interests are teaching and learning software engineering at universities and software engineering didactics. She is with the Faculty of Electrical Engineering and Informatics, University of Applied Sciences and Arts, 96450 Coburg, Germany (e-mail: yvonne.sedelmaier@hs-coburg.de).

D. Landes holds a diploma in informatics from the University of Erlangen-Nuremberg, Germany, and a PhD in economics from the University of Karlsruhe, Germany. After several years in industry, e.g. with Daimler Research, he became a full professor of software engineering and database systems at the University of Applied Sciences, Coburg, Germany. His research interests are in requirements engineering, software engineering education, and data mining. He (co-)authored around 50 papers in books, journals, and conferences in these areas. Since 2012 he is heading the research project EVELIN (e-mail: dieter.landes@hs-coburg.de).

This article is an extended and modified version of a paper presented at IEEE Global Engineering Education Conference (EDUCON2014), held 3 - 5 April 2014, in Istanbul, Turkey. Received 15 July 2014. Published as resubmitted by the authors 14 February 2015.