A Research Agenda for Identifying and Developing Required Competencies in Software Engineering

http://dx.doi.org/10.3991/ijep.v3i2.2448

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

***Abstract*—Various issues make learning and teaching software engineering a challenge for both students and instructors. Since there are no standard curricula and no cookbook recipes for successful software engineering, it is fairly hard to figure out which specific topics and competencies should be learned or acquired by a particular group of students. Furthermore, it is not clear which particular didactic approaches might work well for a specific topic and a particular group of students. This contribution presents a research agenda that aims at identifying relevant competencies and environmental constraints as well as their effect on learning and teaching software engineering. To that end, an experimental approach will be taken. As a distinctive feature, this approach iteratively introduces additional or modified didactical methods into existing courses and carefully evaluates their appropriateness. Thus, it continuously improves these methods.**

***Index Terms*—Software engineering education, academic education, research agenda, grounded theory, didactics, subject didactic software engineering, teaching methodology, soft skills, competencies, research design, curriculum, lifelong learning.**

I. INTRODUCTION

Software is a core part of the modern world. Nearly each aspect of our everyday life is heavily influenced by software. Software can be found on our mobile phones and in refrigerators as well as in cars and heart pacemakers. Consequently, education in software engineering on a university level plays an important role and affects various subjects of study, ranging from informatics through mechatronics and embedded systems to areas with only secondary focus on software such as, e.g., mechanical engineering, applied physics, or business engineering.

Yet, software engineering is not easy to learn, nor can it be taught easily for a couple of reasons.

First of all, software engineering is concerned with building complex systems in a team of developers over an extended period of time for a more or less precisely known group of users. This implies that software engineers need to have various social and personal competencies on top of their technical skills. On the one hand, these personal and social competencies are required to enable them to work in a, possibly interdisciplinary, team of developers. On the other hand, identifying the requirements that users and other stakeholders expect form the software to be built requires elaborate communication skill across disciplinary boundaries. A successful requirements engineer must on the one hand control the technical side and on the other hand understand the interdisciplinary structure of users' thoughts and communication patterns. Furthermore, he must translate the requirements into his own academic discipline software engineering. Yet, it is difficult to adequately represent reality in all its complexity in an academic course. In addition, students can hardly transfer their theoretical knowledge into practical use in a university context.

Secondly, developing software on a large scale involves various roles such as requirements analysts, software architects, or software testers. Each of theses roles has its individual profile with distinct technical, social and personal skills. Consequently, university education in software engineering needs to take into account individual preferences of students for particular roles.

A third challenge lies in the fact that there are no two identical software development projects: if there were no differences at all in the requirements, why should a piece of software be developed a second time if it is already available? Thus, there are no cookbook recipes how to develop software on a large scale that will always work. As a consequence, students need to develop methodological and problem-solving skills which allow them to select, adapt, and combine methods and tools in such a way that they are appropriate for the current project. This is tightly related to practical experience. It is an open issue how students can be given an opportunity to gain that experience in a university setting since there is hardly ever enough time to run large realistic projects.

Finally, information technology in general and software engineering in particular is changing at a rapid pace due to technological and methodological advancements. Therefore, much of the factual knowledge that has been learned and taught during university education will not remain valid throughout the whole professional career. Instead, teaching students how to keep their knowledge and their skills current is a vital part of university education in software engineering. Students need to be prepared for life-long learning.

In summary, these aspects imply that there can hardly be a standard curriculum for software engineering. What comes closest to such a standard curriculum is the Software Engineering Body of Knowledge (SWEBOK) [1] that covers various areas that are relevant for a software engineer. SWEBOK, however, does not take into account to what extent these areas are relevant for which role, nor does is provide any indication which didactic approach might be best suited to learn and teach a particular topic. This is aggravated by the fact that SWEBOK does not

address soft skills, even though they are necessary for a software engineer to apply technical know how successfully. To make things worse, the scope of software engineering education that can be covered as well as applicable didactic approaches are highly constrained by various factors, such as the field of study or group sizes.

In our research, we try to get a better understanding of which factors need to be taken into account and how they affect learning and teaching of particular topics in software engineering. Since there is no solid theory yet, we started a project (Experimental improVEment of Learning software engINeering, EVELIN) to investigate these issues in an experimental fashion. In particular, we are interested in two core research questions, namely

- What are the competencies a software engineer should have?
- Given a particular competence, which didactic approach is appropriate or even best suited to foster it?

In the remainder of this paper, we will outline our research agenda and the chosen approach to pursue it. After this, we will provide a short account of the current status of our research. Finally, we will give a short summary of key issues and an outlook on intended next steps.

II. OVERVIEW OF THE RESEARCH AGENDA

In our research project EVELIN we observe students and their learning processes in software engineering during their academic studies of informatics at Coburg University of applied sciences and arts. Pursuing a bachelor's degree normally takes seven semesters. During this time students need to be equipped with the basic skills required for software engineers since approximately one half of our students leave university with a bachelor's degree rather than enrolling in a master program. If they do the latter, they normally spend another three semesters at university.

In EVELIN we build on already established courses on bachelor and master level that cover various aspects of software engineering ranging from fairly basic issues, e.g. programming, to more advances topics, e.g. model-driven software engineering. Currently, a variety of didactic approaches are already employed in these courses, spanning the range from lectures over practical exercises to final-year projects and research-based learning. These courses provide an opportunity to carefully re-adjust learning and teaching goals and the didactical instruments to achieve these goals. However, there is only insufficient understanding of which didactic approach works well for which topic in software engineering and which environmental constraints affect the effectiveness of didactic approaches in which way. Therefore, we cannot be sure that the selected didactical methods actually perform as expected. Thus, we need to follow an experimental approach to gain more detailed insights into these interdependencies.

Figure 1 presents an overview of our research agenda.

In a first step at time t_0 required competencies of software engineers need to be identified and described. Competencies in this context encompasses soft skills as well as technical know how. The description also includes statements to what degree these competencies are needed or what the precise meaning of the respective soft skills e.g. communicative competence actually is.

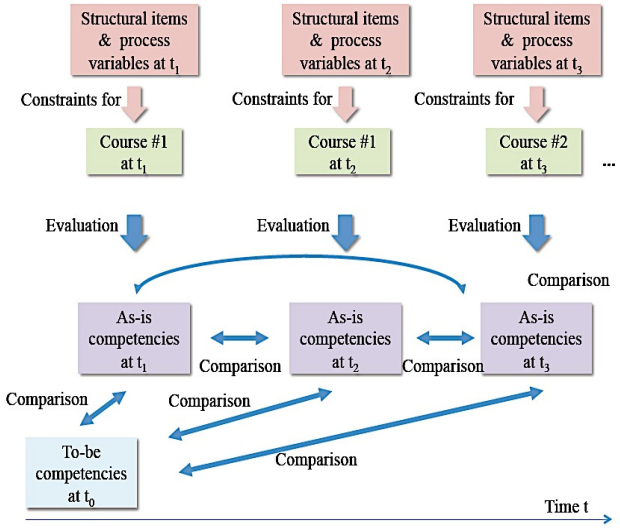

Figure 1. Structure of our research agenda.

Then the actual competencies of our students at a given time t_1 will be collected. This can be accomplished by evaluating their performance with respect to particular competencies in a context of a course, say course #1. To be able to do that properly, we need to establish a baseline of these competencies before students enroll in that course. Competencies after attending the course can then be compared to the baseline. To achieve a deeper understanding we also intend to perform an "as-is" and "to-be" analysis of competencies at this particular time t_1.

Due to the fact that learning processes are influenced by various factors we look at them more closely. Among these factors, process items and structural elements can be distinguished [2] which come in different shapes in the courses we observe.

Based on deficiencies which are identified in the “as-is”, and "to-be" analysis we focus on target competencies that students should acquire and build hypothesis about the influence of structural and process variables we analysed.

Since we aim at improving the levels of competencies that students can gain in a course, we modify predefined structural and process variables we analysed.

Since we aim at improving the levels of competencies that students can gain in a course, we modify predefined structural and/or process variables in a next step in order to achieve such an improvement. Then we evaluate students' competencies in the same, yet modified course at time t_2 and compare the results again with the target competencies and with the reached competencies at time t_1 before changing anything. Thus we expect to gain at least qualitative insights into which variables influence learning and how they do. After gaining a deeper understanding of the influence of process variables and structural items we again modify several of them in a goal-directed fashion and evaluate the results to improve students’ learning outcomes further.

While the analysis mentioned above is based on different groups of students that take the same course at different points in time, we also need to figure out how competencies evolve within the same group of students. To that end, we also compare the acquired level of competencies in an analogue fashion. Again, this comparison encom-

passes an "as-is" and "to-be" analysis of competencies, possibly giving rise to modifications of course #2 with respect to process and structural factors.

We apply this iterative approach to gradually gain a better understanding of which factors influence learning and how they interact. On the basis of such an improved understanding we will be able to systematically adapt structural und process variables and advance students' learning outcomes efficiently.

III. RESEARCH DESIGN

A. General Approach

Our research is based on the Grounded Theory Model [3]. This strategy allows us to discover basic processes which effect change, and we use this methodology to develop hypotheses and theories to better understand learning processes. Software engineering is characterized by a great complexity, requiring various technical competencies in combination with soft skills. Thus, in software engineering, like in several other domains, there are currently no clear cause-and effect relationships among the factors which may influence learning processes. As of now, there are no theories and even less previous knowledge about learning processes in Software Engineering. For this reasons, Grounded Theory seems to be a suitable research strategy that takes these characteristics into account.

Grounded Theory has several characteristic features [3]:

- Grounded Theory aims at building categories by comparing different groups and establishing an understanding of relationships between these categories. Therefore, Grounded Theory is not only a strategy to verify theories but to generate theories. The theory is developed during the research process by building and testing hypotheses. The aim is to explain and to understand learning and teaching software engineering, not only to describe.
- Grounded Theory assumes no predefined research agenda, but rather builds on continuous planning with respect to the next steps to take and data required for these steps. The next steps are determined based on the results achieved so far.
- Theoretical Sampling is a core part of Grounded Theory and means that data collection is controlled by the research interest in the first place and does not primarily focus on data being representative in a statistical sense. Sampling evolves during the whole research process. We analyse qualitative data while collecting, decide which data are needed next while evaluating data, and we stop collecting data once it becomes clear that there will be no further new information. The evolving theory implies which data shall be collected next. For this reason, there are no conclusive statements about the collected data until the research process is finished. So data collection and theory development interact during the whole research project.
- Qualitative methods are used in combination with quantitative ones through triangulation. Qualitative research basically relies on linguistic data, e.g. in the form of texts, while quantitative methods use numerical data [4]. Triangulation means that quantitative data are collected in order to complement and confirm qualitative data and vice versa, thus leading to a more comprehensive view on the area of research [5].

Research projects do not usually start with theoretically deduced hypotheses about the research theme. More usually, at the outset of a research project there are only assumptions and initial knowledge about the research field that need to be structured [6]. Thus, starting from an initial hypothesis, research tries to draw conclusions from previous findings by analyzing their effects (abduction). Through repeated target-directed data collections, preliminary concepts can be developed and specified in interplay of deduction and induction [7].

To achieve reliable results we rely on a mixture of research methods and combine qualitative and quantitative methods. Qualitative analysis builds upon two basic principles, namely the principle of openness [8] and the principle of communication. This means that we investigate in an open und unconstrained fashion and look for previously unknown aspects of learning software engineering. In most cases, we obtain the required data by communication and interaction with involved persons like students or lecturers.

Qualitative analysis often builds on interviews and document analysis. Qualitative research methods aim at a relationship of trust between interviewer und interviewee [8] [9]. For our research it is important to get detailed insights into students' views, opinions, and thoughts and to find out how they learn and what they want or expect to learn. These aspects are often unconscious to students [4]. Yet, a clear picture of these issues is a prerequisite for understanding and supporting learning processes by improving structural and process variables. Similarly, motivational und implicit teaching aspects are unconscious to lecturers. In order to uncover these aspects, we conduct guided interviews face-to-face or on the phone. Guided interviews are semi-structured, loosely following a prepared interview guideline, and allow an open view on previously unknown factors. The interview guideline contains a spectrum of potential questions and focuses on the research themes of interest. It also ensures certain comparability oft the collected data. The interviews are recorded, transcribed, and interpreted. Open questions are employed instead of closed ones to get new information about, e.g., the factors influencing the learning process.

We use qualitative research methods to build and generalize hypotheses and to structure the field of study. We hope that this will finally lead to a theory about learning processes in software engineering.

All in all, in our research we first identify target competencies and collect a broad range of data. Then we set up hypothesis about the influence of structural and process factors that determine learning processes. In a next step we gather as-is competencies at a particular time, modify some influencing factors, and evaluate the competencies again at a later point of time. Then we compare the competencies before and after changing several factors by collecting qualitative and quantitative data and adapt our hypotheses about factors influencing learning processes.

These activities and how we implemented them will be explained in greater detail in the following paragraphs.

B. Specific Steps

1) Identification of Target Competencies

Target competencies include technical know-how on software engineering as well as generic competencies. These generic competencies are not to be confused with general-education subjects such as, e.g., lessons on ethics, but rather refer to capabilities commonly termed soft skills. Soft skills are described by [10] and others. Generic competencies in our research context denote non-technical competencies that a software engineer needs in order to apply his technical know-how and to cope with complex new situations [11]. These competencies are closely related to various personality traits and enable students to act according to the situation.

In order to identify a usable classification scheme for technical competencies, we first took a closer look at existing taxonomies. Several of them concentrate on the cognitive domain or separate the cognitive domain from affective and psychomotor domains, e.g. the taxonomies of Bloom [12], Anderson and Krathwohl [13] or Marzano and Kendall [14]. Some of these taxonomies are strictly hierarchical or overly complex due to multiple dimensions.

As none of the established classification schemes seemed to fit our purpose right away, we decided to develop our own model of description, the EVELIN classification system (see fig. 2 and tab. 1). This taxonomy tries to strike the balance between ease of use and adequate complexity without the necessity for a strictly hierarchical structure.

The EVELIN-Taxonomy consists of the following categories:

a) Remember:

Remember means to know information by heart and to recall it. There is no necessity to understand this information.

b) Understand:

Understand means being able to give a definition of something. The individual is aware of the meaning by herself. Often, understanding is about implicit knowledge which is unconscious.

c) Explain:

If an individual can explain something she has explicit knowledge and does understand the information. To explain something the person must be able to structure information and analyse in a fact-based manner. Normally it is possible to identify advantages and disadvantages. The category "explain" means that an individual is able to evaluate and analyse information in a theoretical way, e.g. with respect to cause-and-effect-relationships, and give reasons for her evaluation or decision.

d) Use:

Use means that a person is able to apply knowledge in a defined and simple context with instructions how to proceed. For "using" it is not necessary to understand something or be capable of explaining it.

e) Apply:

"Apply" includes the capability to use some knowledge and it also requires that an individual understands something on a theoretical level and is able to think about something. Then she can utilise knowledge in complex situations without any help from outside. This requires analysing and evaluating the context before deciding on the most suitable way to solve a problem.

f) Develop:

"Develop" means to create novel solutions or new information and knowledge in a problem domain.

The EVELIN classification system seems to be capable to describe required technical competencies of software engineers in a manageable and understandable way. Furthermore, we view it as a comprehensive instrument for describing learning targets and for planning a curriculum [16]. Although there are some indications, the applicability of the EVELIN classification system to appropriately capture generic competencies, too, still needs to be confirmed on a larger scale.

2) Data Collection

In a first step the EVELIN classification system needs to be instantiated with competencies for each domain, e.g. for informatics or embedded systems. These domain specific competencies shall be compared in a further step. To that end, we used various sources of information. First, we conducted guided interviews with software engineers and managers from several companies. In addition, existing curricula are analysed primarily by document analysis. Curriculum in this context denotes a comprehensive pedagogical concept including didactical aspects as well as methods, contents, and teaching goals. Furthermore, existing references for technical competencies, e.g. the Software Engineering Body of Knowledge (SWEBOK) [1], were examined in detail. Thus we obtained a fairly reliable consistent description of technical and non-technical competencies that are generally needed. This isthe basis to decide which of these competencies can possibly and reasonably be learned in an academic con-

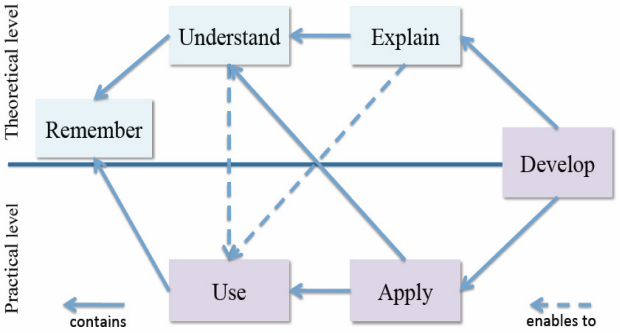

Figure 2. EVELIN classification system to describe technical competencies

TABLE I.
THE EVELIN CLASSIFICATION SYSTEM FOR TECHNICAL KNOW HOW [15] [16] - OVERVIEW

Remember	Recall information and reproduce it
Understand	Capture the sense / meaning of information
Explain	Recognize and understand relationships and analogies between information and explain them in own words (Cause ←→Effect)
Use	Apply in a defined simple context and / or instructed while understanding may not play a role
Apply	Autonomously utilize in a more complex context and /or ability to select and apply the best solution based on the situation.
Develop	Devise new solutions or enhance existing solutions

text. Each lecturer has to define individual measurable teaching targets and to design a specific curriculum with proper didactical methods.

As shown in fig. 1, multiple factors influence learning processes of students. These factors can be classified as process variables, structural items, and outcome [2]. Structural items describe framework conditions that affect learning such as, e.g., the daytime when a class takes place or the number of students in a course. Structure also encompasses issues such as the individual attitudes and capabilities of lecturers, infrastructure, and technical equipment. Process variables describe activities that are necessary to learn, e.g. didactical interactions. Outcome is tightly related to the as-is and the to-be analysis as described in fig. 1. To establish a baseline we conducted guided interviews with a sample of students. The main focus of these interviews was on uncovering factors, which facilitate or complicate learning, and on the perceived learning outcomes from the students' point of view. We also analysed didactical approaches and contents of the courses. By this way we got first indications of factors influencing learning.

Furthermore, the influencing factors of our field of study as well as the as-is and to-be competencies will be analysed in regular intervals. To this end, we conduct guided interviews with students to find out what they learned, how they learned, and what facilitated the learning processes. Uncovering hidden and implicit learning outcomes that students themselves cannot reflect is of primary interest. Similarly, implicit intentions of instructors need to be identified. This is necessary for being able to check if students reached the learning targets the lecturers intended. To that end, we interrogate our students at midterm about the structural framework and the didactical settings of the lessons. For this purpose, we developed a questionnaire which also includes items from BEvaKomp [17] concerning the self-assessment of the students' competencies. The questionnaire also accounts for quantitative data about the technical infrastructure and other structural items and also considers the motivation of students. We evaluate the outcome each semester by analysing the results of examinations of our students and compare this with the estimations of the lecturers.

3) Building Hypotheses

a) Formulating Initial Hypohteses

Following this approach, we expect to be able to come up with suggestions which factors influence the learning outcome. What type of hypotheses will be built and in which way this is accomplished depends on the research method Grounded Theory. This approach works inductively, i.e. we build hypotheses while collecting and evaluating data. One we collected initial data we will get hints to other things to investigate and, as a consequence, we will also collect data concerning these themes. The sample also develops during the research process and is not exactly known at the beginning. Thus, at this point of time we cannot exactly indicate which data will be considered for building hypotheses, nor which hypotheses can be developed based on the collected structural, process, or outcome data. In a first step we "only" start with assumptions. They are gained by concluding from the effects on the previous influencing things (abduction). Fundamental questions include which structural and process factors heavily influence learning processes in a positive way and how they can be promoted.

b) Repeated Empirical Examination of Hypotheses and Theory Building

To validate assumptions and to build categories we modify structural and process variables in a goal-orientated way. We want to find out how and to which degree these variables influence learning processes. So we have to conduct an as-is analysis of competencies before changing anything in order to obtain a baseline. Then we carry out courses in a modified way which is intended to serve the learning goals better. At the end of the course we again collect data concerning the new as-is competencies at this point of time. We compare this result with our baseline data and with the target competencies we want to achieve. There are two benefits in comparing as-is and to-be competencies: On the one hand, we analyse the competencies within one cohort of students over an extended period of time and observe the development of their competencies during their whole academic studies. So we get to know whether and to which degree they reach the target competencies. On the other hand, we compare different cohorts of students attending the same course and compare several instances of results on this specific course over time. By this way we analyse the influencing factors of learning processes within a specific course. From this analysis we get new inputs and indicators how learning processes in software engineering proceed. Based on these new data we are able to build hypotheses that can be tested in the same iterative way and finally lead to a theory of learning and teaching software engineering. The target competencies also will be reviewed at several points in times to adapt them gradually to changing real world requirements.

After going through this cycle for several times, the target competencies are expected to converge to a somewhat stable state (fig. 3) as well as the results from our as-is and to-be analysis (fig. 4).

By this way we get a deeper understanding of the influence of structural and process items on the learning outcome of our students and how to improve and facilitate learning software engineering. Consolidated hypotheses will establish the basis for a theory of learning software engineering.

IV. SUMMARY AND FURTHER WORK

Software engineering is difficult to learn and teach at universities since there is no "one-size-fits-all" curriculum for the subject. The lack of such a generic curriculum is due to the fact that software engineering involves many competencies. These competencies do not only cover technical ones, but also, and equally important, non-technical ones. These competencies need to be present in varying degrees, depending on the particular domain. Applicability of didactic methods depends on a variety of structural constraints and process variables.

In order to be able to improve software engineering education in a systematic and goal-directed manner, a theory is required which competencies are needed in a particular domain and how the development of these competencies is influenced by structural and process issues. As a first step towards such a theory, our research aims at setting up hypotheses on these aspects and at g.

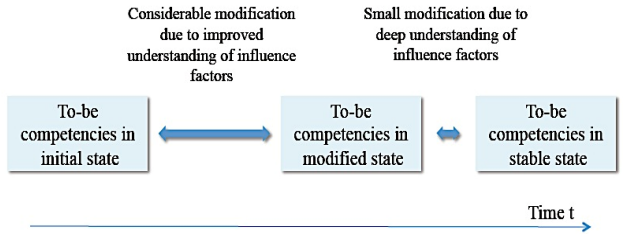

Figure 3. Stabilization of to-be competencies over time

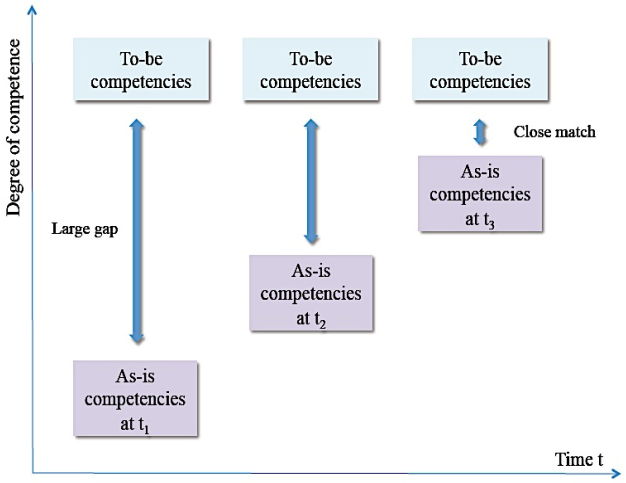

Figure 4. Convergence of intra-group as-is competencies against to-be competencies.

validating these hypotheses experimentally. This theory leads to a subject didactic of software engineerin

At a current stage of the project, we devised a classification system to specify technical competencies. This classification system has already been instantiated with data that were derived form questionnaire-based evaluations of software engineering courses, document analysis, and qualitative interviews with samples of students. Especially the latter two indicated a variety of concrete structural and process variables that need to be accounted for.

We condensed our findings into a first version of technical target competencies of software engineering for informatics. In a next step, required personal and social skills for software engineers need to be added. To that end, we re-analyse already existing interviews with software engineers, but now with a focus on required soft skills. We will widen our scope by analysing needed competencies for related domains, e.g. mechatronics.

We also will take al closer look at the lecturers' aims and their attitudes and personal views on teaching. These are highly relevant factors for designing adequate curricula and for finding proper didactical approaches for each lecturer.

Given the collected data, we expect to be able to build an initial hypothesis on factors influencing learning. This is the basis for conducting a first experimental validation by adapting a particular course, namely a software engineering project at Coburg University, by emphasizing team skills. This course already includes project work but students train their soft skills implicitly without knowing they do. This will change by adding didactical elements to this course in order to make team processes more explicit and conscious.

REFERENCES

[1] A. Abran and J.W. Moore (eds.), Guide to the Software Engineering Body of Knowledge. Los Alamitos, CA: IEEE Computer Society Press, 2004. Available: http://www.computer.org/portal/web/swebok/htmlformat

[2] A. Donabedian, Explorations in Quality Assessment and Monitoring: The definition of quality and approaches to its assessment. Ann. Arbor, MI: Health Administration Press, 1980.

[3] B.G. Glaser and A.L. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research. Chicago: Aldine Publishing Company, 1967.

[4] U. Flick, Qualitative Forschung. Theorie, Methoden, Anwendung in Psychologie und Sozialwissenschaften, 4th ed., Reinbeck: Rowohlt, 1999.

[5] U. Flick, Triangulation - Eine Einführung, 3rd ed., Wiesbaden: VS Verlag für Sozialwissenschaften, 2011. http://dx.doi.org/10.1007/978-3-531-92864-7

[6] H. Legewie, "Qualitative Forschung und der Ansatz der Grounded Theory", available: http://www.ztg.tu-berlin.de/download/legewie/Dokumente/Vorlesung_11.pdf.

[7] U. Flick, E. von Kardorff, and I. Steinke, "Was ist qualitative Forschung? Einleitung und Überblick", in "Qualitative Forschung. Ein Handbuch", Reinbeck: Rowohlt, 2000, pp. 13-29.

[8] P. Mayring, Einführung in die qualitative Sozialforschung - Eine Anleitung zum qualitativen Denken, 4th ed., Weinheim: Beltz, 1999.

[9] H. Hermanns, "Interviewen als Tätigkeit", in "Qualitative Forschung. Ein Handbuch", Reinbeck: Rowohlt, 2000, pp. 360-368.

[10] D. Wilsdorf, Schlüsselqualifikationen. München: Lexika, 1991.

[11] C. Beck, "Kompetenzstudie - Welche Kompetenzen fordern Unternehmen von Bewerbern?", available: http://www.hs-koblenz.de/fileadmin/medien/Koblenz/Betriebswirtschaft/Prof._Dr._Beck/Kompetenzstudie_Final_01.pdf.

[12] B.S. Bloom, Taxonomie von Lernzielen im kognitiven Bereich. Weinheim: Beltz, 1972.

[13] L. Anderson and D.A. Krathwohl (eds.), A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York: Longman, 2001.

[14] R.J. Marzano and J.S. Kendall: The New Taxonomy of Educational Objectives, 2nd ed., Thousand Oaks, CA: Corwin Press, 2007.

[15] S. Claren, "Classification of competencies in Software Engineering Education", unpublished.

[16] S. Claren and Y. Sedelmaier, "Ein Kompetenzrahmenmodell für Software Engineering", Proc. Embedded Software Engineering 2012, Sindelfingen, in press.

[17] E. Braun, Das Berliner Evaluationsinstrument für selbsteingeschätzte studentische Kompetenzen (BEvaKomp). Göttingen: V&R Unipress, 2008.

AUTHORS

Y. Sedelmaier studied pedagogy with a major focus on adult and continuing education at the University of Bamberg. 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) at Coburg University of Applied Sciences and Arts (e-mail: sedelmaier@hs-coburg.de).

D. Landes studied computer sciences at the University of Erlangen-Nuremberg and obtained his doctorate at the University of Karlsruhe. Since 1999 he is Professor in Informatics at Coburg University of Applied Sciences and Arts. Since 2012 he is heading the research project EVELIN (e-mail: landes@hs-coburg.de).

This research is supported by Bundesministerium für Bildung und Forschung under grant no. 01PL12022A. This article is an extended and modified version of a paper presented at the International Conference on Interactive Collaborative Learning (ICL2012), held 26 - 28 September 2012, in Villach, Austria. Received 15 December 2012. Published as resubmitted by the authors 18 March 2013.