A Software Engineer's Competencies: Undergraduate Preconceptions in Contrast to Teaching Intentions

Carolin Gold-Veerkamp
University of Applied Sciences Aschaffenburg, Germany
carolin.gold-veerkamp@h-ab.de

Abstract

Unlike numerous scientific disciplines, the field of engineering has rarely been subject to investigations of undergraduate pre-/misconceptions except for STEM subjects within engineering degrees. When it comes to Software Engineering, some special issues have to be taken into account (e.g. novelty of the discipline and immateriality of the product) that make this discipline hard to teach and learn. Additionally, it requires a wide range of different technical competencies as well as soft skills. As a consequence, the goal is to improve learning by using undergraduates' "right" conceptions as "points of departure" and reduce learning obstacles by facing misconceptions.
This paper is giving some first insights into a quantitative study conducted with undergraduates – before and after instruction – as well as two professors using a questionnaire to rate Software Engineering competencies to elicit preconceptions.

1. Introduction

Software Engineering (SE) has to face already existing and partially growing challenges (e.g. [1–4]: Speed of development, novelty of the discipline, immateriality of the product) that make learning and teaching more difficult. These problems are heightened by the fact that there is a lack of "Fachdidaktik"[1] [5, p. 1f.] in teaching SE.

(Software) Engineers need a lot of technical as well as generic abilities [6] in their role as "Problem Solvers". Considering the competency profile of Software Engineers specified by primary data collections [6, 7], technical competencies are mandatory at least at the level of "understanding" (see Taxonomies: [8, 9], esp. [1, p. 33]).

However, a question arises: What about undergraduates holding misconceptions in Software Engineering?

1.1. Didactic Underpinning

Undergraduates who have serious misconceptions may not be able to acquire a comprehensive understanding and holistic picture of the discipline in the sense of constructivism. From a constructivist perspective, learning is an active and cognitive process in which individuals construct their own knowledge themselves based on prior knowledge, competencies, and experiences that serve as connecting points to which they relate.
Accordingly, learners may not be able to grasp new concepts (during the learning process) and associate them "correctly" with existing knowledge that includes misconceptions.

The scientific ideas/believes of students have been researched for more than a decade [10]. In SE, however, this has been done rarely so far [11, 12] (see Literature Review in [13]).

1.2. Conceptual Delimitation

Preconceptions[2] can have two connotations (cf. [13, p. 710]):

On the one hand, the so-called misconceptions that are "at odds with modern scientific theories" [14, p. 2] and thus can be seen as epistemological learning obstacle [15]; e.g., the common physics misconception that mass is equal to weight (cf. [16, p. 71] and related literature).
On the other hand, if a learner's conception does not contradict the contemporary ideology, it can therefore be seen as "points of departure" [17, p. 6] for new concepts. The idea of an object falling down due to its particular mass and the presence of gravity on

[1] No equivalent engl. exp. (sth. like "discipline specific didactics" or "the way of teaching the discipline")

[2] A lot of publications use the terms "preconception" and "misconception" synonymously, which is not intended in this paper.

URI: https://hdl.handle.net/10125/60217
ISBN: 978-0-9981331-2-6

earth, which are both correct concepts, can be used to learn/teach Newton's 2^{nd} law ($F = m \times a$).

1.3. Research Goal

Instructors should gain awareness and knowledge about possible misconceptions in their discipline in order to be able to consider them when designing as well as implementing teaching/learning arrangements.
Without this knowledge, teaching can neither positively built upon preconceptions, nor can misconceptions as epistemological obstacles be actively addressed with methods such as the Conceptual Change Theory [18] or Didactic Reconstruction [19].

The central research question therefore is: What preconceptions do undergraduates have regarding SE?

On this basis, this paper aims to get a first insight into undergraduate preconceptions in Software Engineering on a rather coarse-grained level of abstraction using a quantitative approach in this pre-study.

1.4. Related Work

As directly related to this work, the publication of Ivins *et al.* [12] has to be mentioned. The authors' objective is to "investigate the preconceptions of first-year students [...]" and to discuss the results "in the context of recruitment and retaining software engineering students" [12, p. 6] using a quantitative approach based on four tasks: Estimating the lines of code, ranking the importance of SE activities as well as skills needed in SE, and evaluating the level of agreement to statements about SE (five-point Likert scale).

The approach in the mentioned publication differs from the one in this paper concerning the following points: A large dataset ($N = 214$) is used, but covers a broad range of specializations (chemical, civil, electrical, mechanical, and Software Engineering). The results of this survey have been analysed for differences between the specializations as well as in contrast to fourth-year SE students, but not the same cohort as the first-year undergraduates [12, p. 7]. The authors did not consult an expert/professional estimation, as they argue: "there is no universally accepted 'gold standard' against which to evaluate the preconceptions of first-year undergraduates" [12, p. 7], as the discipline SE is not mature enough. As the authors pursue the objective to investigate the impact of misconceptions on recruitment, retention, and curriculum design, they do not close the circle and reflect their findings on instruction based on the teaching aims.

Referring to the statement that the authors plan "to design an introductory SE subject [...] to dispel myths and misconceptions about software engineering" [12, p. 11], the observation of Chi shall be noted stating that some misconceptions "are resistant to instructional remediation" [20, p. 161].

2. Survey Methodology

Within this framework, this paper aims to give a first impression of undergraduate preconceptions in Software Engineering on a rather coarse-grained level using a quantitative approach. The research interest in this co-study focuses on engineering degrees (here Mechatronics) covering a limited amount of modules concerned with computer sciences. Thus, the participants successively attend "Informatics I" and "Informatics II" dealing mainly with programming in C followed by "Software Engineering" (4^{th} semester). If they do not decide on a major field of study in information technology (6^{th} and 7^{th} semester), they will leave university with the knowledge about software acquired until the end of semester four. Hence, the module shall give as comprehensive an overview as possible of the discipline. Therefore the teaching aims listed in the module manual [21, p. 19] are:

- Expertise: The students know how to develop software as part of a team and know about the factors due to which software projects can fail. The emphasis is not on programming, but the professional execution of all project phases in the group.

- Skills: The students are able to understand, analyse and systematically implement the client's requirements. They are able to plan a development project, to distribute tasks in the project and to communicate in a targeted way in a team and outside. Also, they are able to carry out the project as well as to test and document the project result.

Based on this holistic approach, the items are therefore partly derived from the Knowledge Areas (KA) of the Software Engineering Body of Knowledge (SWEBOK) [22] to cover professional competencies related to the software life cycle (Table 1). For the most part, these are also part of a competency profile for Software Engineering [6, p. 75].
In addition, two "Context-Sensitive Soft Skills" (C-SSS) and sub-items that emerged from a primary data collection covered in the Software Engineering Body of Skills (SWEBOS) [23] supplement the list of items (D, E, R, and S). They have been chosen, because they have also been elicited in the construction of the already

mentioned competency profile for Software Engineering [6, p. 76] and rated as the most important soft skills.

Table 1. List of Research Items (A-T)[3]

	Items		Source			
Technical Competencies	J	Requirements Elicitation and Management	KA 1	SWEBOK [22]		
	L	Designing Software	KA 2			
	A	Implementation of Code	KA 3			
	H	Testing of Software	KA 4			
	M	Finding All Bugs				
	K	Maintenance and Commissioning of Software	KA 5			
	F	Modifications in Existing Code	KA 5.4			
	Q	Version and Configuration Management	KA 6			
	P	Distributing Software Products	KA 6.6			
	N	Using Defined Process Models	KA 9			
	T	Ensuring Product Quality	KA 10			
	G	Ensuring Process Quality				
	C	Writing and Maintaining Artifacts	KA 11.3.2			
	I	Writing Documentations	KA 11.1.8			
	B	Development of Systems that Contain Code	Realted Disciplines			
	O	Project Management				
C-SSS	E	Working in a Team	KA 11.3.3		(Z)	SWEBOS [23]
	S	Communicating with Various Persons	KA 11.3		(K)	
	R	Performing Meetings				
	D	Performing Customer Meetings				

To get to know the student conceptions on these items, undergraduates have been asked to answer a questionnaire at the beginning of their fourth semester (without any previous instruction) and after the first half of the term. The second date of survey is chosen due to the course design, which is split into seminar and project phase accompanied in parallel with a lecture. Hence, the theoretical input mainly takes place during the first half of the course. After the process of understanding, the gained knowledge should be applied through the second half (performance).
As the focus is on conceptions/understanding – the basis for performance (see taxonomy of learning objectives/competencies [1, 8, 9]) – the survey compares undergraduates' initial conceptions with those elicited after the seminar phase, not their practical usage. Due to the fact that meaningful application of knowledge is based on the need to comprehensive understanding, misconceptions can have wide-ranging consequences when it comes to complex problem definitions. The impact itself is unpredictable as the mental model is a highly complex, interrelated and deeply individual.

The questionnaire design is based on the question "In your opinion, which activities do the work of a Software Engineer include and to what extent?" which has to be answered for the item battery given in Table 1 using a four-point Likert scale (1 – no ... 4 – very great; plus: 0 – not even a part of SE).
Consequently, with the given data, it is possible to get answers to the following questions:

a) How do students rate the items given?
b) How do the responses change after some instruction?
c) How do the instructors rate the items? In other words: Which ratings are intended by the instructors?
d) How well do the undergraduates' answers match the intentions/teaching aims?

It is of interest to compare the undergraduate data with (1) the teaching aim and (2) an expert estimation to detect possible misconceptions. Thus, the decision was made to consult the two professors that are both responsible for this module and share it amongst themselves. As a consequence, the database is very limited, but consists of the persons with a direct influence on these undergraduates and the associated teaching aims.
This shall contribute to get a first impression of the preconceptions undergraduates have and whether they can be used positively in higher education or have to be addressed with respect to the teaching/learning arrangement.[4]

3. Results

Two cohorts (see Table 2) of undergraduates in the Bachelor's degree program in Mechatronics (B.Eng.) took part in the survey (2016 and 2017). Up to the time of the second survey, they have already gained some insights into software process models, requirements management, software project management, version and configuration management, source code documentation and they have also touched on project documentation using artifacts (in classic process models like waterfall or V-shaped).
Before they will have taken part in the module "Software Engineering" they completed two semesters of programming in C.

Table 2. Number of Participants by Cohorts

Cohorts	Expectation Survey	Interim Survey	N
2016	45	46	
2017	38	43	
Overall	83	89	172

[3]Unsorted numbering (A-T) according to the random order in the questionnaire.

[4]This paper can be seen as a study alongside an explorative research using interviews and Grounded Theory Methodology [24]. Besides the difference in qualitative and quantitative approach, this survey is based on competencies the students shall gain exploiting closed questions; which narrow it down, but suitable for a pretest.

3.1. Student Rating of Software Engineering Areas

Before instruction (Fig. 1), the highest evaluations are achieved by (descending order): *E – Working in a Team*, *H – Testing*, *A – Implementation*, and *F – Modifications in Existing Code*. It is conspicuous that three last-named items are closely linked to programming. Moreover, the results scatter ($\sigma = 0.63....1.21$; $CV = 0.18...0.63$)[5], especially when considering the items *C – Writing and Maintaining Artifacts*, *D – Performing Customer Meetings*, *G – Ensuring Process Quality*, *J – Requirements Elicitation and Management*, *Q – Version and Configuration Management*, and *T – Ensuring Product Quality*. Apart from these, *P – Distributing Software Products* has the highest coefficient of variation ($\sigma = 1.08; CV = 1.14$).

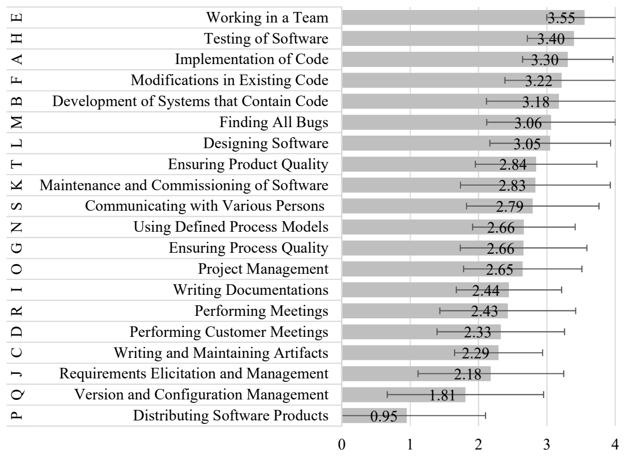

Figure 1. Barchart of Students' Rating Before Instruction (Mean Values and Standard Deviation)

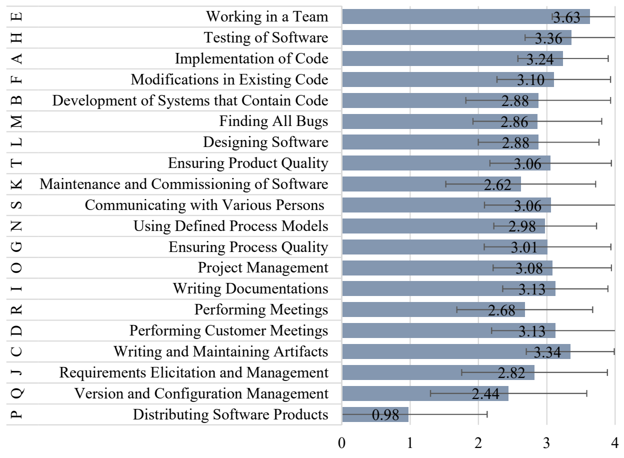

Figure 2. Barchart of Students' Rating After Half the Semester (Mean Values and Standard Deviation)

[5]To be able to interpret the absolute value of σ in a relative context, the coefficient of variation ($CV = \sigma/\bar{x}$) can be used.

The values in Fig. 2 spread again ($\sigma = 0.55....1.15$; $CV = 0.15...0.47$; cf. to Fig. 1); especially at *Q – Version and Configuration Management* and *P – Distributing Software Products*. Once again, the item with the highest $CV = 1.18$ ($\sigma = 1.15$) is *P – Distributing Software Products*, i.e. the participants are uncertain about this item before and after instruction.

The best ratings get: *E – Working in a Team* (unaltered), *H – Testing of Software* (unaltered), and *C – Writing and Maintaining Artifacts* (was on place 16 before). When comparing the two Figures, *A – Implementation of Code* slipped back one place (from 3 to 4) and *F – Modifications in Existing Code* are now on rank 7 (before: 4). *C – Writing and Maintaining Artifacts*, *I – Writing Documentations*, and *D – Performing Customer Meetings* have previously been rated in the lower half. That means that C-SSS and writing tasks seem to increase in value whilst competencies associated with coding depreciate.

Noteworthy, besides the two positions at the bottom in Fig. 1 and the last in Fig. 2, the means of all ratings given are above 2 on the Likert scale.

3.2. Evolution of Student Rating

The following results (Table 3) are based on merely $N = 42$, since for this pre/post study only about half of the returned questionnaires could be unambiguously assigned on the basis of the pseudonyms.

Table 3. Comparison of Undergraduate Ratings Before and After Instruction

Items	Median Rank Expectation Survey	Median Rank Intermin Survey	Δ	z (Wilcoxon signed-rank test)
J	16	10.5	-5.50	-3.187**
Q	17.5	13	-4.50	-3.076**
I	13	9.5	-3.50	-3.448***
C	12.25	9.25	-3.00	-4.163***
D	12.5	9.5	-3.00	-2.969**
G	12.5	9.5	-3.00	-2.104*
N	12	9.5	-2.50	-2.713**
S	10.5	8.5	-2.00	-1.524
T	11	9.5	-1.50	-1.86
O	10	9.5	-0.50	-2.084*
R	12.5	12.5	±0.00	-2.272*
H	6	7	+1.00	-0.599
P	19.5	20.5	+1.00	-0.493
E	5	7	+2.00	-1.000
B	6.5	9.5	+3.00	-1.302
F	6	9	+3.00	-1.553
K	9.5	12.5	+3.00	-0.905
M	7.5	10.5	+3.00	-1.233
A	4.5	8	+3.50	-1.968*
L	6	11	+5.00	-1.302

***. Correlation is significant at the $\leq$ .001 level (2-tailed).
**. Correlation is significant at the $\leq$ .01 level (2-tailed).
*. Correlation is significant at the $\leq$ .05 level (2-tailed).

The biggest changes are visible referring to the median rank concerning (descending order): *J – Requirements Elicitation and Management*, *Q – Version and Configuration Management*, *I – Writing Documentations*, *C – Writing and Maintaining Artifacts*, and *D – Performing Customer Meetings*, whereby all of these show significant differences when considering Wilcoxon signed-rank test.

Based on the hypothesis that the changes (Δ) displayed in the second-last column of Table 3 have resulted from the teaching/learning arrangement "Software Engineering", a glance shall be cast at the experts estimation of the items. Who – in this case – are the two instructors (professors) of this module; to answer the questions: Has this been the instructors' intention? What is working well, i.e. which teaching aims are met? At which points do misconceptions seem to occur?

3.3. Instructors' Rating of Software Engineering Areas

First of all, it has to be noted that there is only one item (*Q – Version and Configuration Management*) with a range in rating of $R = 2$ (marked grey in Table 4). Most items ($n = 11$) are rated identical and the experts ratings vary with a range of $R = 1$ concerning eight items.

The experts rated *D – Performing Customer Meetings*, *E – Working in a Team*, *J – Requirements Elicitation and Management*, *O – Project Management*, and *T – Ensuring Product Quality* as the most extensive elements for the work as a Software Engineer.

Table 4. Instructors'/Experts' Rating

Items	Mean	Range
D	4	0
E	4	0
J	4	0
O	4	0
T	4	0
C	3.5	1
F	3.5	1
G	3.5	1
I	3.5	1
L	3.5	1
S	3.5	1
A	3	0
B	3	0
H	3	0
K	3	0
N	3	0
Q	3	2
R	3	0
M	2.5	1
P	0.5	1

3.4. Comparison of Undergraduates' Conceptions to Experts' and Teaching Aims

When comparing both groups ("teacher" and "learner"), Figure 3 can provide information. For the following items the teaching aim can be considered as met (comparison to interim survey): *C – Writing and Maintaining Artifacts*, *F – Modifications in Existing Code*, *G – Ensuring Process Quality*, *I – Writing Documentations*, *M – Finding All Bugs*, *N – Using Defined Process Models*, *O – Project Management*, *P – Distributing Software Products*, *Q – Version and Configuration Management*, and *S – Communicating with Various Persons*.
In general, the student ratings are usually lower than the expert estimation; both, before and after instruction. Furthermore, the means of undergraduate ratings frequently lie around a value of 3; except for the negative peak at *P – Distributing Software Products* which matches the expert estimation.
To put it into a nutshell, Fig. 3 illustrates that changes – smaller as well as bigger – in the undergraduates' assessment of the items occur due to the present teaching/learning arrangement under investigation.

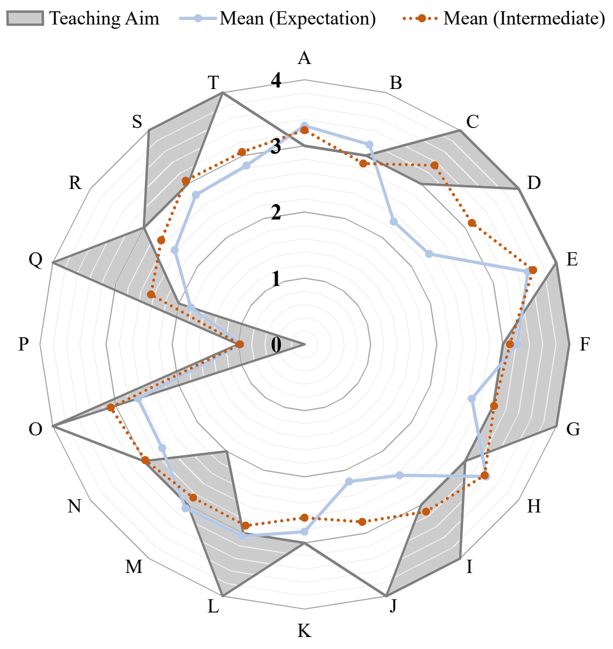

Figure 3. Net Diagram of Student Surveys and Instructors' Teaching Aim Based on Means

3.5. Undergraduates' Level of Concordance

In this Subsection the inter-rater reliability of the students shall be examined. The possibility of checking the trustworthiness should not be waived, as maybe only a few students changed their mind. Therefore, Kendall's coefficient of concordance (Kendall's W; ranging between 0...1) – a non-parametric statistic that can be used to assess agreement among participants – has been calculated, resulting in

$W_{exp} \ (N = 73) = 0.335; p < 0.001$

$W_{inter}(N = 72) = 0.330; p < 0.001$

To interpret Kendalls W, the value area according to [25, p. 257][6] is used leading to a moderate concordance in both surveys. This demonstrates that not all students evaluate the items equally (cf. standard deviation σ in Fig. 1 and 2) and CV in Sec. 3.1, but the level of concordance before and after instruction is almost the same.

4. Discussion: Preconceptions and Misconceptions

The fact that the findings of students' conceptions differ before and after instruction as well as compared to experts itself is not surprising.

The results of the first survey (see Fig 1) display high ratings for activities that are closely related to programming (five items within the top six), e.g. *H – Testing*, *A – Implementation*, *F – Modifications in Existing Code*, *B – Development of Systems*, and *M – Finding All Bugs*. This is barely applicable after instruction (Fig. 2) as well as to the experts rating (Table 4), remarkably. Accordingly, this seems to be an imprinting because of their previously obtained conception; potentially due to the gathered knowledge in programming in the previous two semesters.

E – Working in a Team received the highest evaluation in both student surveys (Fig. 3), equally to the fourth-year undergraduates assessment in [12, p. 9] and the instructor assessment (Table 4).

As Chi distinguishes, "some concepts [...] are resistant to instructional remediation although other, apparently similar concepts are more easily understood" [20, p. 161]. This fact combined with the possibility that instruction itself can cause false conceptions, leads to several potential interpretations when looking at conceptions before and after instruction (see boolean operands in Table 5).

[6] $W \leq 0.3$ – Weak agreement
$0.3 < W \leq 0.5$ – Moderate agreement
$0.5 < W \leq 0.7$ – Good agreement
$W > 0.7$ – Strong agreement

Table 5. Misconception Truth Table

Wrong Concept Before Instruction	Wrong Concept After Instruction	Interpretation
FALSE	FALSE	Good, no work needs to be done here.
TRUE	FALSE	
TRUE	TRUE	Preconception remains.
FALSE	TRUE	Wrong concept came up during instruction.

The entries in the second row can be understood in such a way that teaching/learning is beneficial in contrast to robust misconceptions (row: 3), where action is needed. Equivalent, the teaching/learning arrangement seems to be counterproductive and has to be rethought, when wrong conceptions arise during instruction.

In this regard, Table 6 compares the means of students' expectations and their evolution with the teaching aims ("Professor Survey") resulting in the columns "Δ (Exp−Prof)" and "Δ (Int−Prof)".

Undergraduates seem to have alternative preconceptions concerning the extent to which the following tasks cover the work of a Software Engineer based on the expert opinion (see Table 6, column: "Δ (Exp−Prof)", $>|\ 1.00\ |$): *J – Requirements Elicitation and Management*, *D – Performing Customer Meetings*, *O – Project Management*, *C – Writing and Maintaining Artifacts*, *Q – Version and Configuration Management*, *T – Ensuring Product Quality*, and *I – Writing Documentations*.

After some instruction, several positive changes occur (Table 6, column: "Δ (Int−Prof)" in contrast to "Δ (Exp−Prof), marked light and dark grey), which means that teaching seems to be beneficial here; confer to Table 5.

As already shown in Fig. 3 (Chap. 3.4), the Δ – both, before and after instruction – has a negative value in many cases; i.e. the experts rate these items (top 15 in "Δ (Exp−Prof)" and top 16 in "Δ (Int−Prof)") higher than the students.

However, students seem to have remaining false conceptions concerning *J – Requirements Elicitation and Management*, *T – Ensuring Product Quality*, *O – Project Management*, and *D – Performing Customer Meetings* (Table 6, column: "Δ (Int−Prof). This observation of constancy demonstrates the strength of the mental model [13, p. 711f.] and requires action.

Even changes for the worse can be recognized concerning (Table 6, column: "Δ (Int−Prof)", hatched orange).) at *K – Maintenance and Commissioning of Software*, *L – Designing Software*, *F – Modifications in Existing Code*, and *P – Distributing Software Products*. These items have to be reconsidered regarding instruction.

Table 6. Comparison of Students' Expectations, their Evolution, and the Teaching Aims

	Items	N	Mean Expectation Survey	Mean Interim Survey	Mean Professor Survey	Δ (Exp−Prof)	Δ (Int−Prof)
J	Requirements Elicitation and Management	162	2.18	2.82	4.00	-1.82	-1.18
D	Performing Customer Meetings	168	2.33	3.13	4.00	-1.68	-0.88
O	Project Management	169	2.65	3.08	4.00	-1.35	-0.92
C	Writing and Maintaining Artifacts	169	2.29	3.34	3.50	-1.21	-0.16
Q	Version and Configuration Management	169	1.81	2.44	3.00	-1.19	-0.56
T	Ensuring Product Quality	170	2.84	3.06	4.00	-1.16	-0.94
I	Writing Documentations	169	2.44	3.13	3.50	-1.06	-0.38
G	Ensuring Process Quality	165	2.66	3.01	3.50	-0.84	-0.49
S	Communicating with Various Persons	169	2.79	3.06	3.50	-0.71	-0.44
R	Performing Meetings	169	2.43	2.68	3.00	-0.57	-0.32
L	Designing Software	167	3.05	2.88	3.50	-0.45	-0.62
E	Working in a Team	167	3.55	3.63	4.00	-0.45	-0.37
N	Using Defined Process Models	169	2.66	2.98	3.00	-0.34	-0.02
F	Modifications in Existing Code	171	3.22	3.10	3.50	-0.28	-0.40
K	Maintenance and Commissioning of Software	167	2.83	2.62	3.00	-0.17	-0.38
B	Development of Systems that Contain Code	168	3.18	2.88	3.00	+0.18	-0.13
A	Implementation of Code	170	3.30	3.24	3.00	+0.30	+0.24
H	Testing of Software	169	3.40	3.36	3.00	+0.40	+0.36
P	Distributing Software Products	168	0.95	0.98	0.50	+0.45	+0.48
M	Finding All Bugs	169	3.06	2.86	2.50	+0.56	+0.36

Legend:

▨ Almost the rating of the professors (= teaching aim; Δ(Int−Prof) $\leq$ .2)

■ Changes positively, i.e. Δ decreases ($\geq$ .5)

▩ Changes positively, i.e. Δ decreases slightly ($\geq$ .2 … $<$ .5)

▨ Changes negatively, i.e. Δ increases

"No" changes, i.e. Δ de-/increases $\leq$ .2

Green marked cells indicate that the teaching aim is nearly accomplished at least after instruction.

These analyses show the need for action concerning the biggest issues:

- **Remaining wrong preconceptions** (Table 6, Δ (Int− Prof) ≤ -0.88; cf. Table 5, row 3):
  - *J – Requirements Elicitation and Management*
  - *D – Performing Customer Meetings*
  - *O – Project Management*
  - *T – Ensuring Product Quality*
- **Negative changes** (Table 6, orange hatched; cf. Table 5, row 4):
  - *L – Designing Software*
  - *F – Modifications in Existing Code*
  - *K – Maintenance and Commissioning of Software*
  - *P – Distributing Software Products*

As all participants were students of Mechatronics, the question may arise: Will these results hold up with another cohort from a different university or course of study? Therefore, this research may be claimed to not be representative. In fact, this is one issue faced in detail in the qualitative approach using the Grounded Theory Methodology; here minimum and maximum comparisons are used. Since one goal of a theory grounded in data is to reach theoretical saturation, a quality criterion of such a theory is ensured by the method of continuous comparison. Equally, this quantitative study has the opportunity to widen it and increase its representativeness. Albeit the qualitative research interest is (1) to find and (2) analyse pre-/misconceptions about Software Engineering regardless of university/degree program and establish a first database, to face these challenges by (3) addressing them in teaching/learning arrangements has to be tailored in terms of the single cohort/individual. Hence, this study is a first step focusing on the students, professors, and the module concept of the Bachelor's degree program at our university.

The fact that undergraduates seem to have misconceptions about the extent to which selected tasks/competencies are part of the working life of a Software Engineer by itself seems not to be surprising – at least for a lecturer in this field – however these data are a proof of that.
Of course, the participants in this study are not expected to know or correctly estimate what a working software engineer does. Although, a person holding serious misconceptions may not be able to acquire a comprehensive understanding and a consistent picture of the discipline in the sense of constructivism. Therefore, it is particularly necessary to find appropriate and sustainable ways to address them. The paper does not intend to make a significant contribution toward the didactical challenge in the first place, but takes a step back to (1) detect pre-/misconceptions in order later to be able to (2) analyse/filter for dissemination, strength as well as centrality concerning the discipline [13, p. 712], and to

(3) address them in teaching and learning arrangements.

The whole approach of data collection – no matter whether quantitative or qualitative – has its shortcomings. Namely, posing questions itself leads to a conscious reflection or even stimulates the individual to think about the research content for the first time. It is thus a dilemma that the question per se could trigger a thinking process even though for example no previous conceptions existed. However, otherwise it would not be possible to gain new information and consequently improve learning and teaching from the students' perspective.

5. Summary & Outlook

Through a three-part survey it was possible to get information about undergraduate preconceptions of central Software Engineering competencies on a rather abstract level using the rough granularity of e.g. Knowledge Areas. Further on, the change of their ratings through instruction was analysed.
On the other side, instructors were asked to fill the same questionnaire to be able to use their expert estimation as a reference.
Finally, it could be evaluated whether the undergraduates' answers match the intentions/teaching aims of the instructors. As a result, the participants seem to have misconceptions about the extent to which selected tasks/competencies are part of the working life of a Software Engineer; albeit this is not surprising for lecturers in this field, this is one of the few studies and evidence to date.
Additionally, a qualitative approach is necessary for in-depth findings in order to understand the conceptions of undergraduates using a Grounded Theory Methodology [24].
These results shall then lay the foundation to design a quantitative research approach that "determines" the dissemination of undergraduate misconceptions in SE to complement the qualitative insights [13, p. 711f.].
Therefore the data collection is not concluded yet.

Acknowledgement

The present work as part of the EVELIN project was funded by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant number 01PL17022B. The authors are responsible for the content of this publication. Further thanks is due to Prof. Dr.-Ing. Jörg Abke for making the collection of here published data possible.

References

[1] 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.

[2] J. Hagel, G.; Mottok, *Preamble of the Editors*, pp. v–vi. In: Hagel, G. & Mottok, J. (Eds.): European Conference Software Engineering Education (ECSEE) 2016. Shaker-Verlag, 2016. (ISBN) 978-3-8440-4515-4.

[3] J. Ludewig and H. Lichter, *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken [German]*. Heidelberg: dpunkt.verlag, 3 ed., 2013.

[4] P. Naur and B. Randell, *Software Engineering: Report of a Conference Sponsored by the NATO Science Committee*. Brssel: Scientific Affairs Division, NATO, 7.-11. Oktober 1968.

[5] P. Figas, et al., "Developing Software Engineering Education as a Didactical Discipline in its Own Right," in *Proc. European Conference Software Engineering Education (ECSEE)* (G. Hagel and J. Mottok, eds.), pp. 1–15, Shaker-Verlag, 2014.

[6] C. Gold-Veerkamp, *Erhebung von Soll-Kompetenzen im Software Engineering Anforderungen an Hochschulabsolventen aus industrieller Perspektive [German]*. Springer Vieweg, 1st ed., 2015.

[7] C. Gold, J. Abke, and Y. Sedelmaier, *A Retrospective Course Survey of Graduates to Analyse Competencies in Software Engineering*, pp. 100–106. In: IEEE, Ed.: 2014 IEEE Global Engineering Education Conference (EDUCON). (ISBN) 978-1-4799-3190-3.

[8] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl, *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*. New York: David McKay Company, 1956.

[9] L. W. Anderson and D. R. Krathwohl, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman, 2009. (ISBN) 0-8013-1903-X.

[10] R. Duit, "Bibliography: Students' and Teachers' Conceptions and Science Education," 2009. Version 3, (last access 2017-10-25).

[11] L. A. Sudol and C. Jaspan, "Analyzing the Strength of Undergraduate Misconceptions About Software Engineering," in *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER '10, (New York, NY, USA), pp. 31–40, ACM, 2010.

[12] J. Ivins, B. R. Von Konsky, D. Cooper, and M. Robey, "Software Engineers and Engineering: A Survey of Undergraduate Preconceptions," in *Proceedings. Frontiers in Education. 36th Annual Conference*, pp. 6–11, 2006.

[13] C. Gold-Veerkamp; J. Abke and I. Diethelm, "What About Misconceptions in Software Engineering? A Research Proposal," in *Proc. Global Engineering Education Conference (EDUCON), Athens, Greece* (IEEE, ed.), pp. 709–713, 2017.

[14] J. R. Read, "Children's Misconceptions and Conceptual Change in Science Education," 2004. (last access 2017-10-25).

[15] R. Reuter; F. Hauser; C. Gold-Veerkamp; J. Mottok and J. Abke, “Towards a Definition and Identification of Learning Obstacles in Higher Software Engineering Education,” in *Proc. 9th International Conference on Education and New Learning Technologies (IATED EDULEARN), Barcelona, Spain*, pp. 10259–10267, 2017.

[16] S. Gönen, “A Study on Student Teachers' Misconceptions and Scientifically Acceptable Conceptions About Mass and Gravity,” *Journal of Science Education and Technology*, vol. 17, no. 1, pp. 70–81, 2008.

[17] R. Duit, “Science Education Research Internationally: Conceptions, Research Methods, Domains of Research,” 2007.

[18] S. Vosniadou, ed., *International Handbook of Research on Conceptual Change*. New York: Routledge, 2 ed., 2013.

[19] U. Kattmann; R. Duit; H. Gropengießer and M. Komorek, “Das Modell der Didaktischen Rekonstruktion: Ein Rahmen für naturwissenschaftsdidaktische Forschung und Entwicklung [German],” *Zeitschrift für Didaktik der Naturwissenschaften (ZfDN)*, vol. 3, no. 3, 1997. (last access 2017-10-25).

[20] M. T. H. Chi, “Commonsense conceptions of emergent processes: Why some misconceptions are robust,” *Journal of the Learning Sciences*, vol. 14, no. 2, pp. 161–199, 2005.

[21] University of Applied Sciences Aschaffenburg, “Mechatronik [German],” module manual, 2017.

[22] P. Bourque and R. Fairley, eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK Version 3.0)*. 2014. (ISBN) 978-0769551661.

[23] Y. Sedelmaier and D. Landes, “Software Engineering Body of Skills (SWEBOS),” in *2014 IEEE Global Engineering Education Conference (EDUCON)*, pp. 395–401, 2014.

[24] C. Gold-Veerkamp, “Using Grounded Theory Methodology to Discover Undergraduates' Preconceptions of Software Engineering,” in *Proc. Global Engineering Education Conference (EDUCON), Santa Cruz de Tenerife, Tenerife* (IEEE, ed.), pp. 707–711, 2018.

[25] S. Cafiso, A. Di Graziano, and G. Pappalardo, “Using the Delphi method to evaluate opinions of public transport managers on bus safety,” *Safety Science*, vol. 57, pp. 254–263, 2013.