A Competency Structure Model of Object-Oriented Programming

Matthias Kramer  
Chair for Didactics of Informatics  
University of Duisburg-Essen  
Essen, Germany  
Email: matthias.kramer@uni-due.de

Peter Hubwieser  
Chair for Didactics of Informatics  
TUM School of Education  
Munich, Germany  
Email: peter.hubwieser@tum.de

Torsten Brinda  
Chair for Didactics of Informatics  
University of Duisburg-Essen  
Essen, Germany  
Email: torsten.brinda@uni-due.de

Abstract—Our project COMMOOP aims to develop a competency structure model and appropriate measurement instruments for the field of object-oriented programming (OOP). We started by reviewing existing literature on competency modelling in other subject areas regarding the development methodology as well as the model structures, identified common structural elements, verified, expanded and refined these based on an extensive literature analysis on theoretical and empirical studies on teaching and learning as well as on psychological aspects in the field of OOP. As theoretically derived candidates for potential competency dimensions we identified (1) OOP content knowledge and skills, (2) mastering representation, (3) cognitive processes and (4) metacognitive processes. This theoretically derived model framework was validated based on various competency descriptions in terms of applicability and completeness. For this purpose, we identified competency descriptions related to OOP in 44 computer science curricula and standards from several countries and compared these with our model. Further, we applied it to a list of competency definitions that was extracted by a working group at the ITiCSE 2015 from 14 case studies on K12 in 12 different countries. Finally, the structure model was aligned with the results of a survey among 59 computer science teachers and teacher students on learning difficulties. At the end, it turned out that our proposed model was quite complete already.

I. INTRODUCTION

In Germany, the current state of the art in educational research is to define and measure learning outcomes in terms of competencies. For example, the well-known German Research Foundation (DFG) granted a priority programme, covering 30 projects ranging from theoretical modeling of competencies up to developing instruments for individual diagnostics [26]. Hohler et al. [20] explain, that in this regard multidimensional models of Item Response Theory are suitable to model and measure competencies. The usefulness of such measurements have been discussed by Koeppen et al. [28], e.g. the influence on individual educational decisions or the evaluation of learning outcomes. For an adequate measurement of competencies, usually empirically founded competency models are required. Klieme et al. [25] describe three types of competency models:

1) Competency Structure Models, usually structured by dimensions (e.g. competency areas or competency characteristics) describing the cognitive dispositions that learning individuals need to solve tasks and problems in a specific content or requirement area,

2) Competency Level Models, giving information about the levels or profiles of the described competencies, and

3) Competency Development Models aiming to describe, how competencies will develop over time.

On an international level, a lot of research has been performed in the context of the OECD PISA (Programme for International Student Assessment) studies [41]. Compared to other subjects, competency modeling in computer science education (CSE) is at the very beginning. To our knowledge there has been only one project yet, which intended to develop competency models for CSE and corresponding measuring instruments based on proper empirical research [32]. Yet, the focus of this project was very broad, so that a lot of refining work is needed. Our aim is to refine this model by developing a Competency Structure Model for the introduction of object-oriented programming at high school level. This model should consist of two main components:

1) a set of candidates for (potentially measurable) competencies, and
2) a category system that provides a structure for these competencies.

Presumably, the category system will consist of several separate dimensions that describe different competency factors, which might be relevant for learning object oriented programming. For the development and validation, we have to perform six principal steps:

1) analysis of literature and theoretical derivation of a proposal for the model,
2) collection of competency descriptions candidates for competencies from learners, teaching persons and experts,
3) validation of the model by matching curricula, standards and other sources as well as the found competency descriptions with the dimensions and values of the category system, where necessary adjustment of the model,
4) design and application of item sets that are supposed to measure the competency candidates,
5) analysis of the results applying Item Response Theory [46] regarding the quality of the item sets,
6) dropping of competency candidates with low quality results and finalization of the model.
In this paper, we describe the results of the first two steps of this working plan. Competency aspects relevant for the model are identified (see section II) and suggestions for competency dimensions are elaborated taking into account the structuring results of other disciplines, finishing with a first theoretically derived competency model in section III. For the validation, findings from different empirical studies carried out by the authors are combined in section IV. Furthermore, we give an outlook on the methodology that we will apply to perform the remaining steps of our working plan.

II. BACKGROUND AND RELATED WORK

A. Concept of Competency

By using the term competency we will refer to the definition of Weinert, who defined competencies in his groundbreaking expertise for the OECD [54] as “the cognitive abilities and skills possessed by or able to be learned by individuals that enable them to solve particular problems, as well as the motivational, volitional and social readiness and capacity to use the solutions successfully and responsibly in variable situations”. Regarding measurement, Klieme et al. [25] stated “competencies should be defined by the range of situations and tasks which have to be mastered, and assessment might be done by confronting the student with a sample of such (eventually simulated) situations.” In consequence, the search of such tasks or situations is a very important part of our investigations.

B. Competency Models

Competency models have already been developed and validated empirically in various domains. In the following, we summarize some publications that suggest potential dimensions of our intended model. For this, it should be noted that the characteristic concept of dimension is used with different meanings in these projects, partly in the mathematical sense, partly in terms of the grouping of competencies.

1) Mathematics and Science: The international PISA framework model for mathematics [42] differentiates

- mathematical content,
- mathematical processes, and
- cognitive levels of demand.

The framework for the German PISA test in Mathematics is based on a task model [40], which varies task characteristics in the areas of knowledge and context, solution approach and solution process. A specialized model deals with switching between different mathematical representations in context of problem solving processes [4].

Scientific Literacy is detailed in the PISA framework [42] into

- competencies,
- knowledge,
- attitudes and beliefs.

Lachmayer [31] focused on graph competency in Biology education. She derived a 3-dimensional model consisting of the parts of creating diagrams from given information, gaining information out of a graph or diagram and integrating information by referring to it from a text. The ESNaS model [30] is an interdisciplinary model of the sciences Biology, Chemistry and Physics with the three dimensions

- cognitive processes,
- areas of competency, and
- complexity.

2) Humanities and Arts: The PISA model of reading comprehension [42] includes the text format, reading situation and reading process.

In the DESI project, students’ second language performances were internationally assessed by Klieme et al. [27]. They found the competency categories perception, awareness and production. The RUBiQua-KERK project [29] focused on the psychological construct of religious competency. They derived two main competencies, namely religious participation and religious interpretation. These two competencies were then crossed with several fields of applicability, such as the own religion, other religions and non-religious content.

A similar result can be found in music education, where Jordan and Knigge [24] developed a competency model on perception and contextualization of music. They developed a 4-dimensional model, with the dimensions structured into two groups, perceiving music and applying musical knowledge based on what was perceived.

Frederking et al. [14] investigated, which competencies are necessary to successfully interpret literature in an aesthetic sophisticated way. They derived a 3-dimensional model, consisting of semantic, idiolectal and contextual literary-aesthetic judgement.

3) Computer Science: In 2008, Havenga et al. [19] proposed a model that aimed to measure the increase of students’ abilities in an introductory programming course. They listed the following observable categories:

- cognitive strategies (construction),
- metacognitive strategies (reflection),
- problem-solving strategies (selection), and
- OOP (application).

Yet, at least to our knowledge, no measurement results based on this model were published yet.

The project KOMINA [48] focussed on the definition of competencies for the area of “embedded micro- and nanosystems”. Four competency dimensions were derived, namely

- preconditions,
- development,
- multi-level development, and
- non-cognitive competencies.

The MoKoM project [32] on competencies in the field of system comprehension and modeling resulted in a model consisting of the four dimensions

- system comprehension,
- system development, and
- dealing with system complexity.
4) Problem Solving: For their competency model of problem solving, Fleischer et al. [13] distinguish between the terms
- analytical problem solving, where all necessary information to solve the problem are either mentioned or can be derived from the task itself and
- dynamic problem solving, where a major part of the necessary information to solve the problem has to be derived by interacting with the system in a trial and error process.

A result of their project was the empirical distinction of analytical problem solving competency in three dimensions:
- making decisions,
- analyzing and developing systems and
- identifying mistakes.

5) Synopsis: Despite their amazing diversity, the discussed competency models show some partial accordance. First, most models comprise some aspects of professional knowledge and/or content. Second, several models refer explicitly to content representation issues. Third, all models include an apparent cognitive process aspect that is addressed sometimes very generally, sometimes more specifically as professional or problem-solving processes and in other models quite specialized, for example in terms of judgement or application. Fourth, some models have explicit or implicit references to the problem context. And finally, some models include non-cognitive abilities, for example attitudes, beliefs or awareness.

C. Object-oriented programming

As there are no proposals for empirically founded competency models in computer science (CS) yet, we analyzed literature for research results regarding the factors we have derived in the synopsis of the analyzed competency models, see section II-B5. Yet, as context in its total variety is generally perceived as an indispensable part of any teaching process in CS [9], we don’t assume that our competencies should depend in any way from its specific manifestation. Thus we did not take this factor into account.

1) Knowledge structures: There are numerous works, which deal with structural aspects of OOP, here we can pick out only a few due to space constraints.

Armstrong [2] surveyed literature on object-oriented development and identified eight core concepts of OO, called quarks of object-oriented development, that were identified by the majority of the considered sources. Furthermore she grouped these quarks in a taxonomy:
- Structure: Abstraction, Class, Encapsulation, Inheritance, Object, and
- Behavior: Message Passing, Method, Polymorphism.

Another knowledge structure was proposed by Schwill [50], who adapted the concept of fundamental ideas by Bruner. During the development process he grouped the examples of fundamental ideas under so called master ideas, namely
- algorithmization,
- structured dissection, and
- language.

Recently Mühling et al. [38] investigated the structural knowledge of freshmen of CS students at their university, just before the start of the first lesson. About 50% of the participants had attended a compulsory subject of CS in school over 6 years, while the rest had learned programming outside school, mostly on their own. The students were presented a list of 40 specifically chosen OOP-concepts and asked to draw a concept map from these. The 290 collected maps were combined separately for these two groups by applying the Pathfinder Algorithm (see [49]). In the resulting graphs of both groups, called Concept Landscapes, 6 separated communities could be detected, which suggest the following knowledge dimensions:

1) Data structure (graph, tree, array)
2) Machine (data, working memory, processor)
3) Class & Object structure (object, attribute, association)
4) Algorithmic structure (loops, conditional statement)
5) Representation structure (programming language, syntax, semantics)
6) Execution structure: (statement, program, automaton)

2) Content representation: Obviously, object-oriented concepts are represented in different ways that might also represent different levels of cognitive demands to novices, for example modeling or programming languages (e.g. UML, Java, C#), pseudocode or formal (mathematical) expressions. At least regarding the syntax of programming languages, literature indicates that it is often more than complicated to master.

By investigating roughly 250 students from their beginner course, Garner, Haden and Robins [16] report from several mistakes done by novice programmers. On top, before algorithmic or design problems, there are syntactic problems. It seems that learning a programming language is troublesome for a significant percentage of the novices.

The works of McCall and Kölling [34] point in this direction as well. By analyzing 136 events from 23 sessions, they determined 3 main error categories. Syntactic errors are among semantic errors and logic errors an important identified error category. The latter categories can be traced back to a false understanding of programming logic or programming concepts, which then would be assigned to knowledge about the notional machine or object-oriented principles.

An impressive investigation was made by Denny et al. [10], who could show, that in every quartile from the strongest to the weakest students, uncompilable source code was produced. Even worse was, that weaker students sometimes couldn’t even resolve their syntax problems, which prohibited a working program.

Stefik and Siebert [53] give an overview on the research on syntactical problems that has been conducted so far over the last few years as well as some important results of their own empirical research. Their results give hints, why some students might struggle with programming syntax, e.g. because the perceived intuitiveness varies across word choices and language constructs.
3) Cognitive processes: In the context of taxonomies of learning objectives, several authors have investigated and categorized cognitive processes. Already in 1956, Bloom has presented his well-known taxonomy of cognitive learning objectives [8] that was revised by Anderson and Krathwohl in 2001 [1]. They splitted Blooms taxonomy into the dimensions Knowledge and Cognitive Processes. In the year 2007, an ITiCSE working group conducted a thorough analysis of several taxonomies from the viewpoint of CSE. They proposed a slight adoption of the taxonomy of Anderson and Krathwohl, keeping the original cognitive processes, but splitting the Cognitive Process Dimension further in two subdimensions [15]:

- Interpreting with levels Remember, Understand, Analyze, Evaluate,
- Producing with levels None, Apply, Create.

In its core activity, programming aims to create software, which should provide some intended functionality. Despite an often clear perception of the final product, the way how to produce this software is usually not intuitive at all. Apparently, the process of programming is quite closely related to problem solving.

Polya [44] structured problem-solving processes in different stages, which seem very similar to the results of Fleischer et al. [13], see section II-B4:

- understanding the problem,
- devising a plan,
- carrying out the plan, and
- looking back at the solution.

Winslow [55] transfers the problem solving process into activities during programming. Polya’s problem solving steps can be recognized there, depicted in

- understanding the problem,
- determine how to solve the problem in a general and a formal way,
- translating the problem into a computer language program, and
- testing and debugging the program.

McCracken et al. [37] investigated the programming skills of 216 first-year CS students in a multi-national and multi-institutional assessment. For this purpose, they developed a framework for first year learning objectives. It differs from other frameworks, since it doesn’t focus on content knowledge or principles of a specific programming paradigm but rather on the programming process itself. The assessed categories were

- abstracting the problem from its description,
- generate sub-problems,
- transform sub-problems into sub-solutions,
- re-compose, and
- evaluate and iterate.

4) Metacognitive processes: By investigating factors that predict programming success, meta-cognitive aspects have been shown to be of great importance.

Bergin and Reilly [5] report from a study conducted on 96 students, where they checked 15 factors along with their success on programming. The strongest relationship existed between a student’s perception of their understanding of the module and programming performance. Similar results can be reported by Robins, Rountree and Rountree [45]. They analyzed 472 completed surveys from an introductory programming course and could show, “that the strongest single indicator of success was the grade the student expected to achieve at the beginning of the course”, which is commonly known as self-efficacy.

Hall, Cegielski and Wade observed 139 individuals and found, that theoretical value belief and personality correlated more with programming performance than cognitive ability did [17].

III. THE PROPOSED COMPETENCY STRUCTURE MODEL

A. Theoretical Derivation of the Model

As already explained in section II-B5, we have identified five dimensions that are common to many of the analyzed competency models. Out of good reasons (see section II-C), we have excluded the factor context. Focussing on the remaining four factors, we have performed a literature analysis, aiming to investigate their relevance for our field of OO-programming on the one hand and looking for their inner structures on the other hand (see preceding sections). The results validate that these four factors might be good candidates for the dimensions of our competency model:

1) Knowledge structure,
2) Content representation,
3) Cognitive processes, and
4) Metacognitive processes.

Based on the inspected literature, we will discuss and refine these dimensions in the following sections.

1) Knowledge Structure: As we are focusing on the field of OO-programming, we decided to name this dimension OOP knowledge and skills in our model. Regarding its knowledge structure, we decided to follow the results of Mühling et al. [38], since the works of Armstrong and Schwll can be integrated in their model. Yet, as the relevance of the concept of notional machine has been frequently pointed out, see [11], [51], [6] and [7], we introduced this concept to unify the dimensions 2) Machine and 6) Execution structure of the model of Mühling et al. [38]. Additionally, their factor 5) Representation structure was shifted one level up to form one of the main dimensions of our model and renamed to Mastering Representation. By this way, we ended up in four subdimensions of OOP knowledge and skills: Data structure, Class & Object structure, Algorithmic structure, Notional Machine.

2) Content Representation: The literature analysis gave insight that it seems almost fundamental to master representational aspects to become a competent programmer. The works of Schwll [50] and Mühling et al. [38] (see. II-C1) already also indicated the importance of representation issues. Therefore we introduced a second competency dimension
**Mastering Representation** in our model, although its inner structure seemed not clear enough to fix it already in this proposal.

3) **Cognitive Processes**: Usually programming is never done for itself, but instead to generate a piece of software which serves a specific predefined purpose. Literature gives rise to the assumption that programming processes can generally be characterized as problem-solving processes. Therefore, we decided to include the category Problem Solving Skills as a subdimension of Cognitive Processes in our model. Since it explicitly refers to programming processes, we assume the stages of problem-solving processes by Winslow [55] to form the values of this dimension.

Additionally to this first subdimension, we incorporate a second one for the types of the cognitive processes. For this purpose, we took over the proposal of Fuller et al. [15], because it seems to represent the most elaborated proposal that was developed specifically for CSE, see section II-C3. Also, several of the analyzed competency models of other subjects show a similar separation of deliberating and producing, for example the models of Jordan and Knigge [24] and of the RUBiQua-KERK project [29], see section II-B.

4) **Metacognitive Processes**: As already addressed by Weinert (see section II-A), metacognitive factors like volition, motivation, self-efficacy, perceived understanding or theoretical value belief may play an important role for competency definition and measurement. This is also considered by the presence of such dimensions in many competency models. Yet, in lack of psychological expertise, we will not discuss this dimension in this paper. This might be subject to a future project in collaboration with psychological experts.

**B. The Resulting Model**

As a result of our literature analysis and the deliberations built on its results, we have produced the draft version of a normative competency structure model of introductory object-oriented programming:

1. **OOP knowledge and skills**
   1.1 data structure (graph, tree, array)
   1.2 class & object structure (object, attribute, association)
   1.3 algorithmic structure (loops, conditional statement)
   1.4 notional machine (data, working memory, processor, statement, program, automaton)

2. **Mastering representation (language, syntax, semantics)**

3. **Cognitive Process**
   3.1 Problem solving stage (understanding the problem, determine how to solve the problem, translating the problem into a computer language program, testing and debugging the program)

4. **Metacognitive processes**
   3.2 Cognitive Process Type
      3.2.1 Interpreting (Remember, Understand, Analyze, Evaluate)
      3.2.2 Producing (Apply, Create)

Our theoretically derived model has structural similarities with a model proposed by Havenga et al. [19], see section II-B. In addition, we have incorporated a representation dimension, which was anchored in models of other disciplines (see II-B).

As explained in the introduction already, the purpose of this model is to structure a set of competencies. It consists of four main dimensions and several subdimensions below these. Altogether, all these contribute to a multidimensional competency space. Each competency candidate that is contained therein covers a certain subspace that is limited by fixed values on some dimensions that can be assigned to this competency. For example, the competency candidate “be able to program arrays in Java” might cover a subspace that is limited by following fixed values: 1.1. Arrays, 2 Java, 3.1. Translating, 3.2.1. Understand, 3.2.2. Apply. The remaining dimensions might be covered in total by this competency.

Of course, this model will have to prove its practical applicability in future competency measurements by being able to explain the variance in measurement results. As a first step, we have performed several first empirical steps to start the validation of our model.

**IV. VALIDATION OF THE DERIVED MODEL**

Aiming to validate our proposed model regarding applicability and completeness, we tried to match it to descriptions of competency, educational goals or learning objectives that are described in several curricula, educational recommendations or in country studies of CSE. In addition, we considered the results of a survey among teachers and teacher students on competent behavior in the field of OOP.

**A. Analysis of Curricula and Recommendations**

We analyzed federal (16 states of Germany) and national (Canada, India, UK, USA) curricula as well as recommendations (Code.org, ACM/IEEE curriculum, Computing At School Curriculum, AP CS Principles), altogether 44 documents. Our aim was to identify competency facets in the field of object-oriented programming. This resulted in statements like “Students will demonstrate the ability to create and use instance methods (e.g., constructors, mutators, accessors) in a computer program” [43]. We continued by coding the containing text snippets applying the method of qualitative content analysis according to Mayring (see [33]), combining deductive and inductive coding strategies. First, the dimensions and subdimensions of the proposed model were taken as categories for deductive coding. In the example above about instance methods, the categories Class & Object structure, syntax, semantics and translating the problem into a computer language program could be assigned. Second, some new categories had to be introduced in cases where no suitable categories existed.

Overall, we found 911 relevant competency facets.

Altogether, this validation step demonstrated that most of the detected competency candidates could be integrated in our model by clearly assigning characteristic values on the most relevant dimensions. Nevertheless, some categories had to be added: in the dimension 2 Mastering representation we added
**TABLE I**

**RESULTING CODINGS OF TEACHER SURVEYS**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Concepts</th>
<th>Cognitive Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.</td>
<td>association</td>
<td>recognize, represent</td>
</tr>
<tr>
<td>1.2.</td>
<td>class hierarchies</td>
<td>implement</td>
</tr>
<tr>
<td>1.4.</td>
<td>compiler</td>
<td>explain</td>
</tr>
<tr>
<td>1.2.</td>
<td>constructor</td>
<td>explain</td>
</tr>
<tr>
<td>1.3.</td>
<td>control structures</td>
<td>apply, implement</td>
</tr>
<tr>
<td>3.1.</td>
<td>error messages</td>
<td>analyze, explain, evaluate, interpret, recognize, remove, use</td>
</tr>
<tr>
<td>1.4.</td>
<td>garbage collector</td>
<td>understand</td>
</tr>
<tr>
<td>2.</td>
<td>implementation</td>
<td>locate</td>
</tr>
<tr>
<td>1.4.</td>
<td>in-output</td>
<td>implement</td>
</tr>
<tr>
<td>1.2.</td>
<td>inheritance</td>
<td>explain, master</td>
</tr>
<tr>
<td>1.2.</td>
<td>interface</td>
<td>apply</td>
</tr>
<tr>
<td>1.3.</td>
<td>local variables</td>
<td>implement</td>
</tr>
<tr>
<td>1.2.</td>
<td>method</td>
<td>implement (for different classes, nested, return values)</td>
</tr>
<tr>
<td>1.2.</td>
<td>modelling</td>
<td>explain importance</td>
</tr>
<tr>
<td>1.2.</td>
<td>modularization</td>
<td>explain</td>
</tr>
<tr>
<td>1.2.</td>
<td>object</td>
<td>explain instantiation, distinguish from class, reference</td>
</tr>
<tr>
<td>1.3.</td>
<td>parallel programming</td>
<td>implement</td>
</tr>
<tr>
<td>1.4.</td>
<td>processes</td>
<td>partition</td>
</tr>
<tr>
<td>3.1.</td>
<td>program</td>
<td>find requirements, overview</td>
</tr>
<tr>
<td>2.</td>
<td>program code</td>
<td>interpret</td>
</tr>
<tr>
<td>1.2.</td>
<td>reference attributes</td>
<td>distinguish</td>
</tr>
<tr>
<td>2.</td>
<td>syntax</td>
<td>apply</td>
</tr>
<tr>
<td>1.4.</td>
<td>variable</td>
<td>explain, declare, change values, switch values,</td>
</tr>
</tbody>
</table>

**TABLE II**

**EXEMPLARY ITiCSE WG COMPETENCY DESCRIPTIONS**

<table>
<thead>
<tr>
<th>Nr</th>
<th>Students should be able to ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>use a formal language to describe an object in an algorithmic way</td>
</tr>
<tr>
<td>37</td>
<td>explain the execution of a sequence of simple instructions of typical machine language</td>
</tr>
<tr>
<td>60</td>
<td>program overloaded methods and constructors in practice</td>
</tr>
<tr>
<td>121</td>
<td>create classes in general, followed by the concepts of inheritance and polymorphism</td>
</tr>
</tbody>
</table>

**TABLE III**

**ITiCSE COMPETENCY DESCRIPTIONS OF SUBDIMENSION 1.2**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>associations</td>
</tr>
<tr>
<td>manipulate</td>
<td>attributes</td>
</tr>
<tr>
<td>understand</td>
<td>automatic garbage collection</td>
</tr>
<tr>
<td>create</td>
<td>Classes</td>
</tr>
<tr>
<td>use</td>
<td>concept of object state</td>
</tr>
<tr>
<td>use, apply</td>
<td>encapsulation</td>
</tr>
<tr>
<td>program</td>
<td>instance variables and accessors</td>
</tr>
<tr>
<td>understand</td>
<td>lifespan of an object</td>
</tr>
<tr>
<td>overload, create, call</td>
<td>methods</td>
</tr>
<tr>
<td>implement</td>
<td>object-, class-, state-models</td>
</tr>
<tr>
<td>program</td>
<td>objects</td>
</tr>
<tr>
<td>use</td>
<td>overloaded methods and constructors</td>
</tr>
<tr>
<td>apply</td>
<td>reference types</td>
</tr>
<tr>
<td></td>
<td>sub- and superclass, inheritance, and polymorphism</td>
</tr>
</tbody>
</table>

*Code Conventions and Documenting & Maintenance, in the subdimension 3.1 Problem solving skills we added Evaluating different strategies.*

**B. Conduction of surveys**

During the year 2015, we have organized a large in-service training for about 200 high school teachers and teacher students on Master level. Among them, we conducted a survey on learning difficulties for programming novices. We asked them: “According to your experience, what are the most difficult steps in learning to program?”. The idea was that difficult learning steps might potentially represent competencies. From 59 teachers or teacher students answered this question, we received 123 relevant descriptions of potential competencies. The coding results in the dimensions 1 and 3.2 are listed in Table I.

At the end, we were able to match all coded answers into our existing model, hence our proposal seemed complete already.

**C. Results of the ITiCSE-WG on country reports**

Recently, the ACM journal *Transactions on Computing Education (TOCE)* published two special issues on CSE in Schools [36], [35], containing 13 extensive case studies of 11 countries or states. Additionally, a pilot article had been printed in advance [22]. In July 2015, an international working group at the ACM ITiCSE conference in Vilnius analyzed these case studies [23], aiming to summarize all this information to answer several research questions. One of these questions asked, which goals or competencies were intended to be achieved by K-12 CSE in these countries or states. The working group found 247 competency descriptions covering the different areas of CSE [23]. 119 of these competencies were related to programming in any way and thus seemed relevant to our project. Table II displays some randomly picked examples.

After coding the 119 relevant competency descriptions according to our proposed model, we found that all could be integrated. Thus, our model seemed quite complete once again. As an example, Table III shows the codes of the competency descriptions that could be assigned to fixed values in the subdimension 1.2 *Class & Object structure.*

**V. Future Work and Conclusion**

According to the literature on competency modeling (see II-B) the process of item construction is of significant importance for the empirical validation of the theoretically derived model. At this point, it is apparent that suitable tasks are required as blueprints for items. Out of these deliberations, we had analyzed more than 300 programming tasks from different sources. Two sources were written specifically for OO-Programming in high-schools: a widely used textbook [21] and an “official” recommendation for competency-oriented tasks [52]. From the university context, we selected the CS1 course of the highest ranked CS department in the respective state. The course comprised a lecture and additionally a mandatory practical course in object-oriented programming. We analyzed the tasks of this practical course.

Finally, we chose a frequently used book for self-learners [12] about object-oriented programming. We had to restrict the tasks of this source to those that were in accordance with the scope of our educational context. Afterwards, we
had categorized the selected programming tasks regarding the addressed subject matter knowledge elements (concepts) and the cognitive processes that are required to solve them. The coding was performed by two researches in parallel, coordinated by a common coding phase at the beginning. To assess intercoder reliability, 68 tasks were coded by both coders. This resulted in 1482 congruent codings.

The resulting category system comprises 113 different concepts, specifying knowledge elements and 70 different operators for cognitive processes. Yet, it turned out that there were many synonymous verbs among the cognitive operators, e.g. transform and modify. Thus we introduced 16 upper-level categories for the 70 operators. These results will enable us to find suitable tasks for many combinations of a cognitive operator (e.g. apply) and a knowledge concept (e.g. repetition).

Due to the complex structure of a competency, it can’t be measured with one single item, as explained e.g. by [18]. Instead, usually about 10 items are applied for each competency. To get more items out of a few task blueprints, we will vary our tasks according to the 11 types of programming assignments derived by Ruf et al. [47], for example write a program (or a part of it), adjust or extend the given solution to the problem, consider the effects of executing the given code or transform the given code according to the given prerequisites.

In consequence, the next step will be to construct sets of about 10-12 items by this way for several competency candidates, which will be selected as narrow as possible to get acceptable results. How this could look like, was demonstrated very recently by Mühling et al. [39] for the competency candidate “apply control structures”.

After construction, the item sets will have to be tested by conducting appropriate pretests among more than hundred test persons. The goal of these pretests is to investigate the quality of the item sets regarding homogeneity, which means that all items of one set are measuring the same psychometric constant, for example a certain competency. An appropriate methodology for this purpose is Latent Trait Analysis, which is explained very instructively in by Bartholomew in Chapter 8 of [3]. Another methodology that will be considered is the nonparametric analysis as applied by Mühling et al. in [39]. In any way, we will need some hundreds of test persons for our pretest.

It is planned to carry out the process of item construction and validation with the help of educational psychologists to ensure high quality of item examples and corresponding construction guidelines as well as a transformation from a competency model to a psychometric measurement model.

REFERENCES


