The Y Model - Formalization of Computer-Science Tasks in the Context of Intelligent Tutoring Systems

DOMINIC LOHR, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
MARC BERGES, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
MICHAEL KOHLHASE, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
DENNIS MÜLLER, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
MAX RAPP, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Tasks are central elements in computer-science classes irrespective of the mode of delivery – online or classic face-to-face teaching. They appear in different roles; as learning objects that help acquire competencies, assessment instruments for measuring learning success, or practicing specific skills. However, as with any educational technology, proper handling and well-planned selection are crucial for successful teaching. Correspondingly, the requirements that tasks have to fulfill are wide-ranging: they should promote specific learning outcomes, be appropriate to address desired cognitive dimensions and be tailored to the student to keep motivation and learning success high. To enable an on-demand, outcome-oriented, and individualized selection of tasks in intelligent tutoring systems, we need to formalize tasks in all dimensions. A detailed review of different models for cognitive processes, knowledge dimensions in education, tasks, and errors leads to the Y-model, a model for formalizing tasks in computer science and forms the basis for an individual, competence-oriented integration of tasks in Intelligent Tutoring Systems.

CCS Concepts: • **Applied computing → Computer-managed instruction**; **Interactive learning environments**; **Computer-assisted instruction**; **E-learning**; • **Computing methodologies → Semantic networks**.

Additional Key Words and Phrases: y-model, computer-science tasks, intelligent tutoring systems, semantic annotation, task formalization, task selection, task modeling

ACM Reference Format:
Dominic Lohr, Marc Berges, Michael Kohlhase, Dennis Müller, and Max Rapp. 2022. The Y Model - Formalization of Computer-Science Tasks in the Context of Intelligent Tutoring Systems. In . ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The pandemic showed an increased need for online available educational resources. Different approaches were taken into account when transforming old lecture materials from classic face-to-face learning into blended-learning or other online formats. While some video-recorded their classes, did others switch the learning paradigm [38]. In addition, the number of students in STEM subjects is increasing, and so is the industry's need for well-educated computer scientists. However, it is not only a large number of learners that challenges teachers; the group of learners is also becoming increasingly heterogeneous in terms of prior knowledge, social background, and learning performance [3].

The high number of students in STEM programs, combined with fundamentally scarce resources, creates a strong demand for systems that support or even (partially) automate grading and feedback [13, 15], as well as systems that support handling heterogeneity [10].

Modern learning environments such as Intelligent Tutoring Systems (ITS) try to meet this heterogeneity and enable adaptive learning. Here, adaptive learning means that learning content is tailored to the learner and appropriate tasks are provided at the appropriate time, with valuable individual feedback [20]. However, some learning platforms (e.g., for learning to program) only offer a predefined and fixed sequence of learning tasks and provide simple feedback [17, 21]. A predefined task sequence is not equally suitable for individual learners who can differ in many facets, such as prior knowledge, personality, learning speed, and many more [31]. A central aim of an adaptive learning environment must therefore be to select learning tasks tailored to the learner to enable a maximum learning outcome and keep motivation high.

However, the selection of tasks is a central professional competence of teachers [28], and tasks show a considerable variety. So, to support learning, tasks should promote specific competencies, not be too difficult but challenging, in the best case, tailored to the student so that motivation is promoted - to name a few requirements from a long list. In addition, tasks should be selected so that the necessary prior knowledge is activated and the learning objective can be achieved. Teachers often choose tasks intuitively based on experience or recommendations.

On the other hand, students' answers given to tasks often vary greatly. Responding to each individual is impossible in most cases due to a lack of resources. Talking to experienced tutors reveals that many answers have specific patterns and can be addressed with the same feedback. Likewise, it is not easy to decide which task to choose in order to be able to successfully teach or test a certain learning content in a certain complexity.

So, one of the major challenges of teaching is to provide learners with appropriate tasks at the appropriate time. The provision of tasks can have different reasons. They can be used to check learning success, teach new content and concepts, or even practice them. Tasks are a very complex entity and are, therefore, often the subject of research [33].

In the context of Intelligent Tutoring Systems (ITS), task selection is related to the learners' educational biography. Algorithms for task selection are developed and compared with decisions made by experts [30, 31]. Kicken et al. investigated how to design on-demand education to, e.g., promote self-direct learning skills so that students can self-select tasks. In some systems, tasks contain meta-data with information about the difficulty, skills, and knowledge needed to complete that task. This information should help learners self-select the tasks that best fit their learning process. However, even when learners are provided with a portfolio in addition to this meta-information, which informs them about their level of performance, not all learners succeed in selecting appropriate tasks. Sometimes the process of self-selecting tasks leads to additional stress, high mental effort, and demotivation [22]. Self-Selecting tasks also bear the risk that students overestimate themselves or that only easy tasks are selected, and the learning effect remains low. Both inevitably lead to the learners' demotivation or the fact that specific learning goals are not achieved. Especially in CS1 courses, it is advantageous if an experienced teacher or learning system makes this selection. Here, a limitation to adaptive system-controlled task selection instead of learner-controlled task selection is made. Nevertheless, the ability of students to select tasks themselves is essential, but it can lead to unnecessary cognitive load, especially at the beginning of the learning process.

Finally, it is necessary to formalize all involved dimensions of the construct *task* in order to be able to incorporate them into the decision-making process. A task always consists of knowledge components as well as cognitive processes that are necessary for successful processing. The knowledge dimension is discussed in Section 2.2, while the cognitive dimension of tasks in Section 2.3. In addition, Section 2.4 introduces answer classes - a technique of clustering answers into classes, which helps to further develop automated feedback and correction. Section 3 of this paper presents an example of a possible implementation of the task-modeling presented in Section 2.

2 REPRESENTING TASKS WITH THE Y-MODEL

The elements connected to tasks must be formalized to better integrate tasks into Intelligent Tutoring Systems or other educational technology. First, a task model is built to illustrate the interdependencies between those elements. The central element of the suggested task model is the task itself - represented by its description. The upper part ("arms") illustrates the interaction of knowledge elements on the one hand and cognitive processes on the other. The knowledge elements and their interdependencies are modeled using *knowledge graphs* based on LoGraph modeling theory by Kohlhase et al. [24]. The learning taxonomy of Fuller et al.[14], explicitly developed for computer science, is used to model cognitive processes.

The lower part of the task model ("foot") adds *answer classes* to the tasks. Here, (possible) answers are classified based on similar answer patterns or adaptive feedback. Additionally, answer classes enable selection processes of tasks based on prior knowledge and competency goals, as well as depending on answers given by students from previous tasks. In addition, specific error patterns, such as those proposed by Zehetmeier et al. [41] or Berges et al. [7], can be integrated into answer classes and, for instance, infer possible causes of the errors by referring to a corresponding user model. In addition, the answer classes enable addressing suitable feedback or hints on how to proceed.

For an overview of the task model's components and their underlying theories, see Figure 1. The model is displayed as a "Y" and called Y-model in the following text.

The components of the Y-model mentioned above are explained in the following sections, accompanied by a running example task (see Figure 2).

2.1 Task

In terms of competency-based teaching, it is part of teachers' pedagogical knowledge [16] to develop and/or select tasks on which students can acquire and enhance the targeted competencies [12][25][34][36][40]. Tasks have various functions: they are suitable for imparting knowledge, can be useful to practice what has already been taught, and are important for checking learning success.

2.1.1 Definition of task. The term *task* is widely used and has several different meanings. While a *task* is defined in dictionaries as *a piece of work to be done, especially one done regularly, unwillingly, or with difficulty* [1], in the context of teaching, most definitions emphasize a prompt and its processing by the student [35]. However, common to almost all definitions is a description of a target state to be achieved.

Here, the term *task* is understood in a general sense, which means that even a simple request to read a text and remember the content is already understood as a task.

The only limitation is that a task always involves a knowledge element and a cognitive process.

According to [33] a task is defined as an instruction (*to do*) to transform a given state (*given problem*) into a desired final state (*expected solution*). In the initial and final states, knowledge is represented and transformed into each other using a cognitive process (see Figure 3).

In general, tasks can be concisely formulated (see Example 2.1) so that particular prerequisites are only implicit in the task description because they are only known in the context of the course but not written down explicitly. For example, in the context of programming tasks, it is not explicitly stated with each task in which programming language the

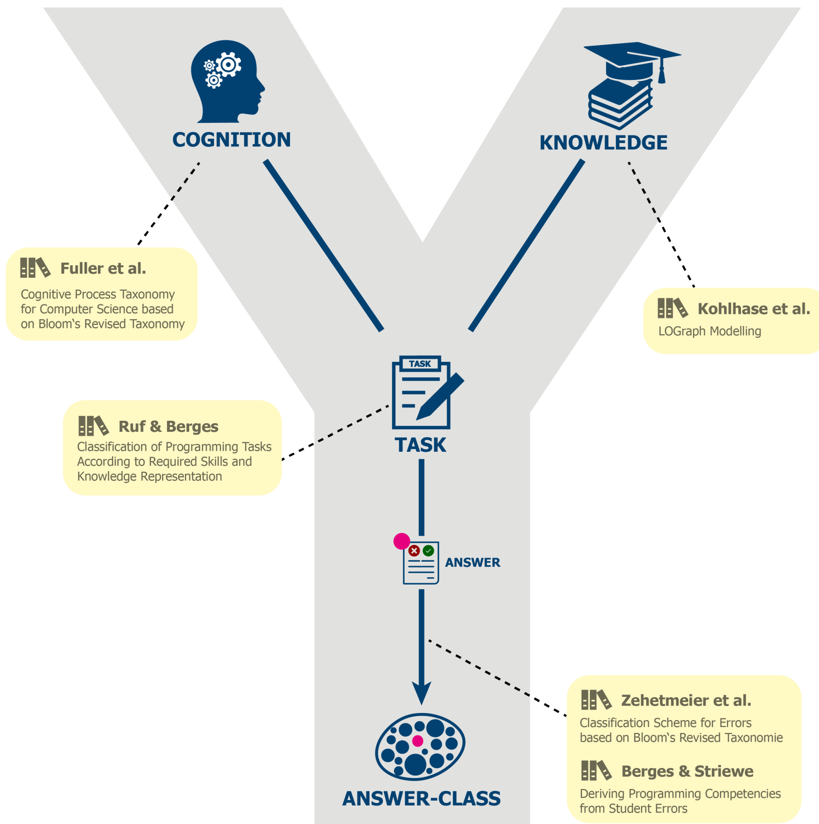

Fig. 1. The Y-model

Task

Given the following empty class body `Homework3` in Java:

```java
public class Homework3 {
    // Code here
}
```

Implement a method `minSearch` which receives an array of integers as input and returns the minimum of those numbers.

Fig. 2. Running Example

program is to be written or whether certain restrictions (like the prohibition to use the Java-API) are given. Nevertheless, this might influence possible answer classes and, in particular, expected errors.

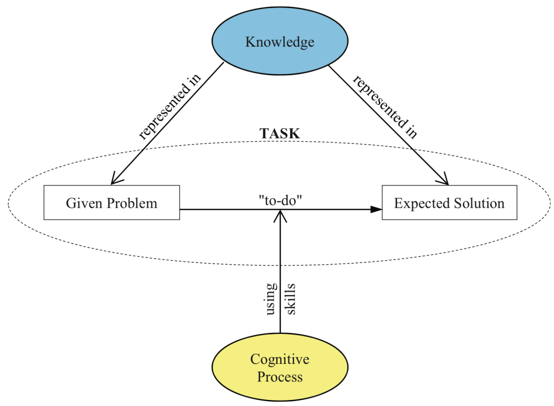

Fig. 3. The roles of knowledge and cognitive process while solving a task [33]

Example 2.1. Write a method that outputs the minimum..

given state	empty class body `Homework3`
desired state	class `Homework3` with a method `minSearch` that returns the minimum value of a given array of integers

Table 1. given and desired state of working example

However, in all cases, they can be reformulated to consist of an initial and final (desired) state. For instance, Example 2.1 can be reformulated to a task description with more detailed information about the given state (see Figure 2 and Table 1).

2.1.2 Classification of tasks. To get a deeper understanding of the task itself, classification schemes can describe the setting, the context, or, as mentioned above, the involved cognitive processes and knowledge elements. The latter is crucial for connecting the conceptual knowledge of a domain, the learners, and the tasks. For that reason, they have a direct representation in the model. Nevertheless, different other models provide essential elements.

The general task model for selecting tasks by Maier et al. [27] proposes seven categories. The first category classifies tasks on the addressed type of knowledge based on the corresponding dimension of Bloom's revised taxonomy [5]. The cognitive process is listed in general categories like reproduction, close transfer, wide transfer, and problem-solving. Furthermore, the number of dependent knowledge elements indicates the perceived difficulty of a task, just like the focus or the complexity of the task description. The motivational aspects of tasks are covered by the relation to the students' reality and the number of representations.

The classification scheme of Ruf et al. focuses on the type of representation of the given task and the expected solution, as well as on the transition process. They analyzed the foremost programming tasks and identified eleven different types according to [33]. Both category systems agree that less information in the task description about how to get to the desired state leads to more open tasks.

2.2 Knowledge Dimension

There is no universal definition of the term *knowledge* across domains. For example, philosophical-epistemological treatments of knowledge usually require known propositions to be *true*. In education, there is a substantial distinction between competencies and knowledge. While the former focuses on the observable outcome and often refers to the definition of Weinert [39], the latter is described as the representation of information in memory [4].

In contrast, for our purposes *knowledge* is anything amenable to techniques developed in the field of *knowledge representation*. Knowledge, in this sense, neither needs to be correct, believed, or even propositional, nor does it need to be associated with any cognitive abilities.

In other words, in the context of the Y-model, *knowledge* is understood pragmatically as any information contained in the course materials. In addition, the understanding includes tasks that have to be represented in a form actionable by a machine that can provide added-value services, including the curation and selection of tasks, as well as providing feedback to students' answers to a given problem.

Thus, an essential aspect of the knowledge component of the Y-model is *structural*: how does a piece of information relate to the broader body of knowledge? What must a learner already know to understand it or to complete an associated task? Which knowledge elements does a given problem *evaluate*?

In the field of knowledge representation, knowledge elements are often represented in *knowledge graphs*. Knowledge graphs provide a structured representation of a given body of knowledge with nodes representing fundamental knowledge elements and edges relations *between* knowledge elements. The exact nature of knowledge elements and relations can vary widely (for a recent overview, see [18]).

For our purposes, we use the sTeX system [24, 29], a LaTeX package and associated software for augmenting document fragments with semantic annotations. Besides being standard LaTeX, which is likely familiar to many teachers in STEM fields, sTeX documents can be converted to HTML and imported and processed by Mmt [32], a software system and Java/Scala API for generic knowledge management services. Mmt can subsequently serve the HTML as an *active document*, integrating various semantically informed added-value services, including interactive ones. Both Mmt and sTeX organize individual knowledge elements as *declarations*, that document fragments can be annotated with. Declarations are collected in *modules*, which may import and extend other modules analogously to object-oriented programming. This allows for representing knowledge as an interconnected *theory graph* of modules (see Figure 4). In particular, sTeX has previously been used to semantically annotate thousands of pages of university course materials using declarations from reusable libraries with > 2250 concepts and is extensible concerning annotation types.

2.3 Cognitive Dimension

To successfully complete a task, the existence of pure knowledge is not sufficient. Specific cognitive processes are required to retrieve the knowledge or to apply it to a problem. However, the concrete cognitive processes involved in solving a task remain a black box for the teacher.

Modeling cognitive processes in the context of teaching has been a topic in research since the 1950s. Choosing the right educational taxonomy is critical for developing learning objectives and assessing learning outcomes [14]. A variety of taxonomies already exist to describe and classify cognitive performance. In higher education, Bloom's taxonomy [9], its revised version by Anderson & Krathwohl [5], and the SOLO taxonomy [8] are often used. Bloom's taxonomy consists of the six categories *1) Knowledge*, *2) Comprehension*, *3) Application*, *4) Analysis*, *5) Synthesis*, and *6) Evaluation* (see left part of Figure 5), which are arranged hierarchically. Bloom assumes that cognitive processes on a

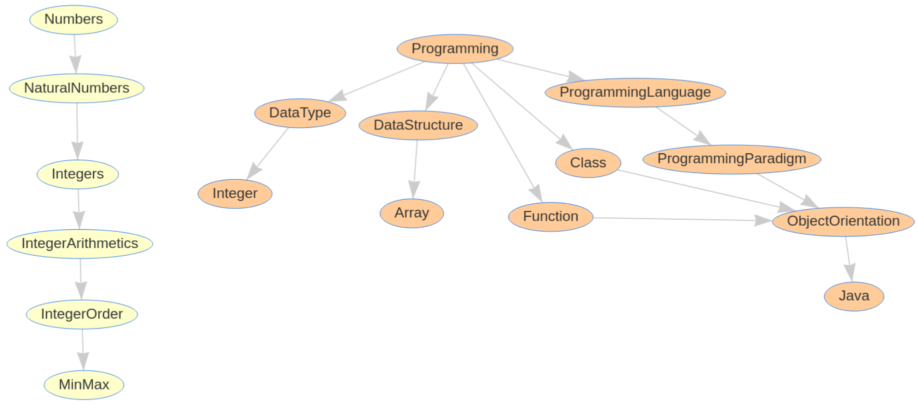

Fig. 4. A Theory Graph of Concepts Used in the Running Example

higher level depend on those from lower levels. Whether this assumption is reasonable, however, is already discussed by Bloom himself [9]. Anderson & Krathwohl picked up the critique and rearranged the cognitive process dimension (see right part of Figure 5). Additionally, they introduced a new dimension related to knowledge [5].

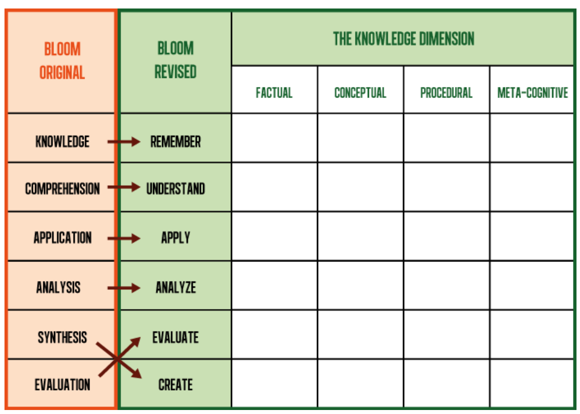

Fig. 5. The comparison of the original taxonomy by the revised taxonomy for cognitive domain and the taxonomy table [2]

Johnson and Fuller conducted a study to examine the extent to which existing learning taxonomies (especially the previously mentioned) are appropriate in the context of Computer Science and how they are used in Computer Science Education (e.g., design of courses, teaching materials, assessments, and the analysis of student responses to exercises or measuring student progress) [14][19]. They examined the weaknesses of existing taxonomies from a CS point of view and concluded that they are only suitable to a limited extent. For instance, they identified a principal weakness of Bloom's taxonomy (and other hierarchically arranged taxonomies) in that when used to assess practical subjects such as programming, the levels do not appear to be well ordered. In a programming context, it seems easier to apply knowledge to solve simple problems than to describe that knowledge [19]. The same phenomenon was observed by Berges et al. [6]. Based on these results, Fuller et al. developed a taxonomy explicitly applied to the CS context and used in the Y-model for modeling the cognitive dimension.

In their taxonomy, they separated the six levels of Bloom's taxonomy into two dimensions (*Producing* and *Interpreting*) to remove the strict ordering (see Figure 6). The first dimension (horizontal axis) represents the ability to understand

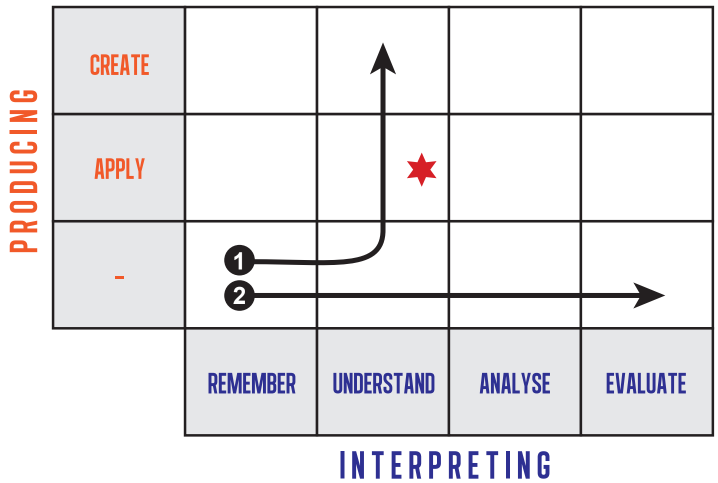

Fig. 6. Two dimensional adaptation of Bloom's taxonomy by Fuller et al. [14]

and interpret an *existing* product. In contrast, the vertical axis represents the ability to design and build a *new* product. The lowest levels are placed in the lower left corner, and it is understood that students traverse them from left to right and from bottom to top, respectively. For example, according to Fuller et al., it is not possible to start refactoring code if there is not yet a certain level of competence for application. A significant advantage of this matrix taxonomy is that different learning paths can be considered, which is especially useful when generating *Guided Tours*. Guided Tours are mini-courses consisting of all prerequisite knowledge leading up to some specific learning goal that can be tailored to individual learners. A path in the horizontal direction of the matrix means that the learner acquires only theoretical competencies (see path 2 in figure 6), whereas a movement in the vertical direction results in the expansion of practical competencies (see path 1 in Figure 6). When applying the taxonomy to the working example, however, some difficulties arise:

When categorizing a task (like the presented example) in terms of the cognitive component, it can only be concluded which cognitive process must have occurred with a certain probability when a particular task was successfully processed. Nevertheless, the actual cognitive process always depends on previous experiences and prior knowledge of the learners and can not be annotated in advance.

[19] confirm that even experts often disagree on which cognitive category to place a task in. For this reason, the Y-model only annotates which cognitive process is targeted in connection with the knowledge elements when processing a task. It does not generally determine which task requires which cognitive processes to solve it successfully. The latter would only be possible by modeling the user and is the goal of future work.

As with the knowledge dimension, for the annotation of the cognitive dimension, (an extension of) sTeX is used. Since the taxonomy of Fuller et al. is a two-dimensional matrix, the noted cognitive elements E also consist of a tuple of two: $E \in$ (PRODUCING × INTERPRETING). Tasks in which program code is written are always APPLY or CREATE. In which dimension of INTERPRETING the task is located depends on whether, for example, the algorithm used in the program is already known and is only written down (or copy-paste) from memory or whether the algorithm is applied to solve another problem.

2.4 Answer Classes

A given answer to a task provides a valuable source of information in the context of teaching. In a sense, answers result from applying cognitive processes to the learning content addressed in the task. Thus, an answer provides evidence for the teacher about the level of students' performance. The selection of the following tasks in the learning process and individual feedback also heavily depend on the learner's answer to a task (a more detailed example is given in Section 3.1). For these reasons, a task formalization must include modeling its possible manifestations of answers.

This part corresponds to the foot of the Y-model. Given answers are grouped into *answer classes*, which can be understood as a set-theoretic propositional form. Let R be a possible response to a task and B be an observable description of the state of this task. We say that R is the element of the answer class AC_x if R meets all the requirements of description B. Formal:

$$AC_x = \{R \mid R \text{ meets all the requirements of description B}\}$$

We always understand requirements that determine an answer class to be objectively observable (e.g., cognitive processes are not considered in answer classes because they are not objectively observable).

In the simplest case, a task contains only two answer classes: the class of answers that meet all specifications of the final state (A) and the complement of that class ($\neg A$).

Table 2 shows some examples of answer classes for the task *minSearch* (see Figure 2):

ID	Answer Class
AC_1	$\{R \mid R$ is written in the programming language JAVA$\}$
AC_2	$\{R \mid R$ compiles with JAVA-Compiler Version X$\}$
AC_3	$\{R \mid R$ includes a syntax error$\}$
AC_4	$\{R \mid R$ includes an algorithm that correctly identifies the minimum of an array of integers$\}$
AC_5	$\{R \mid R$ includes a while-/for-loop$\}$
AC_6	$\{R \mid$ use of `System.out.println` instead of `return`$\}$
AC_7	$\{R \mid$ uses the Java-API$\}$

Table 2. Example of answer classes for a task

Since the task did not explicitly require that the answer be solved using a loop and the use of the Java API was not explicitly prohibited, many different approaches provide an accepted solution (see Examples 2.2, 2.3 and 2.4).

Example 2.2.

```
import java.util.Arrays;
public class Homework {
    public int minSearch(int[] list) {
        Arrays.sort(list);
        return list[0];
    }
}
```

In Example 2.2, the passed array is first sorted in ascending order (using the Java API), and then the first element of the sorted array is returned.

Example 2.3.

```
import java.util.Arrays;
public class Homework {
    int min = list[0];
     public int minSearch(int[] list) {
          return Arrays.stream(list).min().getAsInt();
     }
}
```

The answer in Example 2.3 also uses the Java API. This time, however, the minimum is computed directly.

Example 2.4.

```
public class Homework {
    int min = list[0];
     public int minSearch(int[] list) {
          for (int i = 1; i < list.length; i++){
               if (min > list[i]) {
                    min = list[i];
               }
          }
     return min;
}
```

The task is successfully processed in Example 2.4 without using the Java API. A for-loop is used to iterate through the array, and a variable `min` is used to locate and output the minimum value of the given array.

The answer class that meets all specifications of the expected state would consist of the following intersection of answer classes: $A = AC_1 \cap AC_2 \cap \neg AC_3 \cap AC_4$
If the expected state is further restricted so that the use of the Java API is not allowed, then $R \notin \mathrm{AC}_7$. The expected solution could also be explicitly required to contain a while-/for-loop ($R \in \mathrm{AC}_5$).

A significant advantage of grouping answers into answer classes is the opportunity to provide feedback more efficiently. Now, it is only necessary to address the answer class instead of every individual answer inside the class. This circumstance reduces the effort dramatically. However, the challenge remains in identifying the respective answer classes for a given answer. A diagnosis is necessary to decide which set of answer classes a given answer is an element of. In the context of programming tasks, this can be done by static and dynamic code analysis. For example, static code analysis makes it possible to determine if the Java API has been used (by parsing for import, for example). Using unit tests makes it possible to decide if the given answer is semantically correct ($R \in \mathrm{AC}_4$). However, answer classes do not only help to identify correct answers. For example, answers based on the same error cause can also be grouped together and addressed with valuable feedback or hints on how to proceed.

3 Y-MODEL IN ACTION

After introducing the model based on a theoretical derivation, implications for creating and selecting tasks are presented.

The running example introduced in Section 2.1 is used as an initial task description. Simple tasks such as multiple-choice or fill-in-the-blanks questions could be generated automatically from the marked-up lecture notes. For example, a multiple choice question can be generated from the knowledge graph by asking students to select the correct definition of a concept from a range of options drawn from other concepts in the graph, definition variants of the concept, and frequent misconceptions about the concept. Likewise, fill-in-the-blank texts can be generated by simply omitting marked-up elements from the lecture materials. However, in most cases, the task description is more complex. Hence, creating a Y-model requires manual annotation.

Before being able to start formalizing the task according to the Y-model, authors must first clarify the context in which the task is created. Notably, the same task description may correspond to different specific Y-models: for example, the task could be used in a course on *algorithms and data structures* aiming to test the student's grasp of iterating over sequences, or be given in the context of sorting algorithms; or it could appear in a CS1 course focusing on the application of control structures, yet a different use case would be an introduction to Java for experienced programmers who merely need to learn the particular syntax and style conventions of the language. Depending on this context, the two "arms" of the Y-model will differ concerning the cognitive and knowledge dimensions for the task's specific *objective*.

Regarding the presented example, the prerequisite concepts for annotating the task are already declared in sTeX modules, and the objective is to test a student's ability to use *arrays*.

3.1 Creating Tasks

Once authors have chosen a proper context, they can begin semantically annotating the task itself. The bare LaTeX of the task-description above might look something like in Figure 7.

```
Given the following empty class body \texttt{Homework3}:
\begin{listing}[language = Java]
    public class Homework3 {
        // Code here
    }
\end{listing}

Implement a method \texttt{minSearch} which receives an array of integers as input and returns the minimum of those numbers.
```

Fig. 7. The Running Example in LaTeX

Conversely, Figure 8 shows a fully sTeX-annotated version of the same snippet.

sTeX provides a custom environment for *problems*/exercises (opened in Line 1), behaving like modules – i.e., they can import other modules and declare new knowledge elements. Our particular example task depends on the modules for the programming language Java, integer and array datatypes, and the minimum function on sequences of integers (Lines 2–5). The modules imported form the theory graph representing the task's *learning context* from Figure 4.

The declarations in these modules can subsequently be used to annotate their occurrences in the text (Lines 6–16, names of declarations highlighted in green, referenced via \symbolname).

Up to this point, the annotation process is independent of the context the task is used in: knowledge elements can and *should* be reused from the existing theory graph to minimize redundancy.

However, we cannot infer from the mere task description what the educational goals of the task are, which requires additional, context-dependent annotations.

3.1.1 Objective. The first step is to determine the *objective* of the task. The objective specifies the learning goal about which we want to gather information using the task. Since carrying out the task also affects a transformation of the

```
\begin{problem}
    \import{Java}
    \import{datatypes?Integer}
    \import{Array}
    \import{MinMax}
    Given the following empty \symbolname{class-body} \texttt{Homework3}:
    \begin{listing}[language = Java]
        public class Homework3 {
            // Code here
        }
    \end{listing}

    Implement a \symbolname{method} \texttt{minSearch} which receives an
    \symbolname{array} of \symbolname[integer]{integers} as
    \symbolname{Function?input} and returns the \symbolname{MinMax?minimum}
    of those \symbolname[number]{numbers}.

\end{problem}
```

Fig. 8. The Running Example in LaTeX, Semantically Annotated (sTeX syntax slightly simplified for clarity)

user's knowledge, the convention is that the objective aims at the state of the learning goal *after* the user has completed the task. According to [5], a learning goal consists of a knowledge and a cognitive dimension.

In general, setting the objective is a modeling decision that specifies which task the user is believed to complete when working on the assignment. For instance, the objective is the tuple $\langle [understand, apply], array\rangle$ (see red star in Figure 6) which is reflected by inserting an additional annotation `\objective{understand}{apply}{array}`: it is assumed that the user demonstrates and/or acquires the competence of understanding and applying arrays by completing the task. This assumption might be cross-validated by further tasks (for example, to rule out that the user just memorized the solution to this task without any deeper understanding).

3.1.2 Preconditions. However, a task does not only carry information about its *main* objective but also about implicit learning objectives inherent to the task. They are modeled as *preconditions* that enable the user to complete the task or even grasp the description itself. Like objectives, preconditions are tuples of sets of cognitive abilities and knowledge elements. In the simplest case, the knowledge dimension of a task's preconditions is simply the theory graph induced (via topological closure) by the imported modules. However, preconditions can naturally be categorized in *necessary* and *sufficient* ones. For example, a lack of understanding of the minimum function will likely lead to errors when trying to solve the task. As such, $\langle [understand, -], minimum\rangle$ is a necessary precondition for this task. On the other hand, mastery of sorting algorithms in Java is not necessary to solve the task but certainly suffices. Many incorrect answer classes will be associated with a missing necessary condition. For instance, the class of answers that fail to return 1 for the array $[1, 1, 2]$ might indicate a lack of understanding of the minimum function - and additionally, perhaps the misconception "*The minimum is defined to be unique. Therefore an array with two smallest elements has no minimum.*"

On the other hand, correct answer classes will often correspond to sufficient preconditions. For example, a solution that uses a tournament algorithm allows us to infer the user's mastery of this method.

In a first step the focus is on necessary preconditions, which we specify by adding a new annotation `\precondition{understand}{}{minimum}`. Additionally, by default every concept mentioned via `\symbolname` is assumed to be a precondition with respect to the cognitive dimension *remember*. Naturally, a precondition itself subsumes *some* implied preconditions on dependent concepts – e.g., it is impossible to understand the concept of a *minimum* (on integers) without also, to some degree, understanding the concept of the *order* of integers – and, indeed, integers themselves.

3.1.3 Answer Classes. As described in Section 2.4, answer classes consist of a collection of similar answers concerning one or more criteria. There are, in principle, two approaches to creating answer classes for a task: during creating/developing the task (top-down) and clustering given answers to answer classes (bottom-up).

Every answer class consists of a unique identifier, a description/criteria, and (optional) a representative answer in that class. In the running example annotating the answer class AC_6 looks like this:

```
\answerclass{AC6}{..\aclasses\ac6.tex}{-10} %Prints a value instead of returning it
```

An analysis of the answers is necessary for each answer class to assign the answers to the appropriate answer classes. Several general ITS techniques to analyze answers already exist, such as *Model Tracing*, *Constraint-based modeling*, *Data Analysis*. Furthermore there are Domain-specific techniques such as *Automated Testing*, *Basic Static Analysis*, *Intention-based diagnosis* and *Program transformation* (for a detailed overview and explanations see [21]). In the provided example, basic static code analysis could be used to determine whether the `return` keyword is used in the code or `System.out.println`. Additionally, it is even possible to use Automated Testing (e.g., unit-tests) to check if the correct result is printed to the console, which means that the algorithm is correct and the answer is an element of AC_6. Obviously, most answer classes (e.g., the class of answers with a syntax error AC_3) are global and apply across several tasks.

Another application of the answer classes is systematized grading. By adding an additional argument, certain properties of an answer can be assigned a specific score.

A possible example would look like this: An answer that is semantically correct ($R \in AC_4$) should start with 50 points. If it contains syntactical errors ($R \in AC_3$), 10 points should be deducted. The task is evaluated with zero points if the Java standard library is used ($R \in AC_7$).

This leads to the following annotation in the task:

```
\answerclass{AC3}{..\aclasses\ac3.tex}{-10} %includes syntax errors
\answerclass{AC4}{..\aclasses\ac4.tex}{+50} %includes an algorithm that correctly identifies the minimum of an array of integers
\answerclass{AC6}{..\aclasses\ac6.tex}{*0} %Prints a value instead of returning it
```

Answer classes are also a valuable tool to update *user models*: If an ITS has some model of a user's prior knowledge and abilities, we can use answer classes to update this model based on their answer to a given problem. In a simplified case, it can be assumed that a user's cognitive state is modeled as a matrix of knowledge elements and cognitive processes. Suppose the given answer is in all three answer classes AC_4, AC_5, and AC_6 (e.g., method with a correct algorithm using a for/while-loop but prints to console instead of returning the result). In that case, we can update the user model correspondingly by increasing our confidence in their understanding of control structures but possibly decreasing our confidence in their understanding of *return* statements.

The answer classes alone can not determine the precise *cause* of a user's mistakes. For that reason, such a user model can be considered to generate personalized *feedback classes* as combinations of user model states and answer classes.

3.2 Selection of Tasks

In education, there are a variety off criteria to select the appropriate task for a learner:

1) The intended learning goal,
2) The prior knowledge of the learners,

3) What answers were given to a previous task,
4) Which tasks are already known by the learner,
5) The personality of the learner

– just to name a few. Notable, for many of the above criteria, information about the learners must be available. However, user modeling is not part of the Y-model, so we assume that this information is known to the teacher or that a user model already exists in an ITS.

Tasks are usually selected to promote or test specific competencies. Using the Y-model, each task contains at least one or more *objective*. If a certain learning content is addressed at a specific cognitive level, it can filter for tasks aimed at this learning objective. In addition, requirements for prior knowledge can be set and compared with the existing tasks. Ideally, this is done individually by matching the user model.

However, selecting single tasks is not the only benefit when modeling tasks with the Y-Model. It is also possible to generate sequences of tasks (e.g., Guided Tours) fitted to a given user model. For example, if the user model does not match the prior knowledge of a task, a sequence of tasks can be offered beforehand, which addresses missing competencies as objectives. These sequences can be dynamic and change individually, even during sequential processing, depending on answers (answer classes). In an adaptive system, each learner would have a different sequence of tasks based on given answers and the individual user model (which aims at a certain level of competence).

Referring to the running example, if it turns out that the learner has a syntax error in their solution ($R \in AC_3$), which is due to a knowledge gap (e.g., they does not know that in Java Syntax, a Semicolon is needed after each statement), a possible follow-up task can be given to close this knowledge gap (see Figure 9). The new task (Task 2) could include information about Java syntax and an elementary code example in which semicolons must be inserted. When the given answer compiles correctly ($R \in AC_2$), the next task (Task 3) can be selected, and so on.

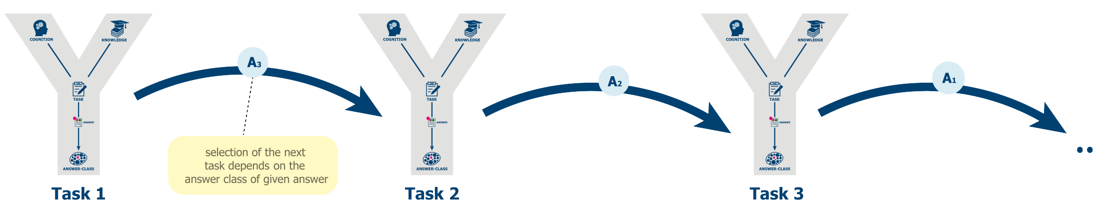

Fig. 9. Selection of tasks based on answer classes (and user model)

4 CONCLUSION AND FUTURE WORK

Using existing models and current research, the knowledge and the cognitive dimensions of tasks were cast into a model. In addition, answer classes were modeled, significantly expanding the possibilities for individual feedback and task selection processes. The Y-model was developed in the context of programming tasks, so the selection of individual models and literature research was made with a focus on programming tasks. However, the Y-model can be applied to other areas as well. In this case, the underpinning models like the one for the cognitive process dimension may be needed to be reevaluated. The presented technique for semantic annotation of course materials (see sTeX in Section 2.2) is limited to using LaTeX documents. Nevertheless, the theoretic model is suitable for other contexts as well.

Further development efforts are made to integrate annotations into all types of course materials (e.g., Powerpoint slides, pdf-documents). Additionally, experiments with alternative software (such as word processors) are ongoing.

In general, the semantic annotation of course materials using the Y-model acts as an interface between the teacher's pedagogical content knowledge and the adaptive systems such as Intelligent Tutoring Systems. This allows multiple valuable services such as generating individualized feedback and updating a system's model of users' knowledge and cognitive state. Although the focus is at the moment specifically on tasks related to computer science, the model itself should generalize well to other areas with minor modifications.

Although the model allows the selection of tasks based on prior knowledge and learning objectives, it does not yet offer the possibility to evaluate the cognitive load of tasks. Cognitive load is crucial in planning task sequences or generating guided tours. For this reason, one of the following steps is to integrate Cognitive Load Theory [11, 37] into the model.

Formalizing tasks alone is insufficient to enable individualized learning in an adaptive system. The Y-model focuses mainly on task-specific criteria for task selection. However, it also depends on learner-specific criteria for which task is appropriate to achieve a particular learning level and promote selected competencies. Each student has an individual educational biography [23]. In order to be able to provide information about the learner and integrate it into the decision-making processes of the system, a user model is necessary and part of future research. Course materials have already been semantically annotated according to Y-model specifications. Using these materials and some exemplary expressions of user models, procedures will be developed for generating learner-tailored tasks and guided tours from the information provided. The results should then be compared and evaluated with the decision of experts.

Generating course materials is not the only focus of future efforts. Also, the generation of adaptive feedback and hints is planned as future work. Research shows that systems mainly generate simple feedback that only addresses the error itself but does not address the cause of the error [21]. Valuable feedback, however, addresses the root cause of an error [26]. Combining the Y-model with a user model, we will explore techniques to infer the cause of errors from errors.

The presented model is a theoretical underpinning for different applications of integrating tasks in adaptive educational technology like Intelligent Tutoring Systems. The ongoing work will focus on the further development of formalizing educational approaches and on developing systems that apply the proposed model.

REFERENCES

[1] 2011. *Cambridge business English dictionary.* Cambridge University Press, Cambridge.

[2] 2012. *Encyclopedia of the Sciences of Learning.* Springer US, Boston, MA.

[3] 2022. Record numbers of students choose Computer Science A Level in 2022. https://www.bcs.org/articles-opinion-and-research/record-numbers-of-students-choose-computer-science-a-level-in-2022/

[4] John R. Anderson. 2005. *Cognitive psychology and its implications* (6th ed.). Worth Publishers, New York.

[5] Lorin W. Anderson and David R. Krathwohl. 2009. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives.* Longman, New York.

[6] Marc Berges, Andreas Mühling, and Peter Hubwieser. 2012. The Gap Between Knowledge and Ability. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research - Koli Calling '12.* ACM Press, New York, 126–134.

[7] Marc Berges, Michael Striewe, Philipp Shah, Michael Goedicke, and Peter Hubwieser. 2016. Towards Deriving Programming Competencies from Student Errors. In *4th International Conference on Learning and Teaching in Computing and Engineering (LaTiCE).* IEEE Press, Los Alamitos, 19–23.

[8] John B Biggs and Kevin F Collis. 1982. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome).* Academic Press.

[9] Benjamin Samuel Bloom. 1956. *Taxonomy of educational objectives: The classification of educational goals.* McKay and Longman, New York and New York and London.

[10] Dino Capovilla, Marc Berges, Andreas Mühling, and Peter Hubwieser. 2015. Handling Heterogeneity in Programming Courses for Freshmen. In *3rd International Conference on Learning and Teaching in Computing and Engineering (LaTiCE).* IEEE Press, Los Alamitos, 197–203. https://doi.org/10.1109/LaTiCE.2015.18

[11] Paul Chandler and John Sweller. 1991. Cognitive Load Theory and the Format of Instruction. *Cognition and Instruction* 8, 4 (1991), 293–332. https://doi.org/10.1207/s1532690xci0804$\backslash${_}2

[12] Gemma Corbalan, Liesbeth Kester, and Jeroen J.G. van Merriënboer. 2008. Selecting learning tasks: Effects of adaptation and shared control on learning efficiency and task involvement. *Contemporary Educational Psychology* 33, 4 (2008), 733–756. https://doi.org/10.1016/j.cedpsych.2008.02.003

[13] Karol Danutama and Inggriani Liem. 2013. Scalable Autograder and LMS Integration. *Procedia Technology* 11 (2013), 388–395. https://doi.org/10.1016/j.protcy.2013.12.207

[14] Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L. Lewis, Donna McGee Thompson, Charles Riedesel, and Errol Thompson. 2007. Developing a computer science-specific learning taxonomy. *SIGCSE Bulletin* 39, 4 (2007), 152–170.

[15] Georgiana Haldeman, Andrew Tjang, Monica Babeş-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D. Nguyen. 2018. Providing Meaningful Feedback for Autograding of Programming Assignments. In *SIGCSE'18*, Tiffany Barnes, Daniel Garcia, Elizabeth K. Hawthorne, and Manuel A. Pérez-Quiñones (Eds.). ACM Association for Computing Machinery, New York, NY, USA, 278–283. https://doi.org/10.1145/3159450.3159502

[16] Peter Hubwieser, Marc Berges, Johannes Magenheim, Niclas Schaper, Kathrin Bröker, Melanie Margaritis, Sigrid Schubert, and Laura Ohrndorf. 2013. Pedagogical Content Knowledge for Computer Science in German Teacher Education Curricula. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education.* ACM, New York, 95–103. https://doi.org/10.1145/2532748.2532753

[17] Johan Jeuring, Hieke Keuning, Samiha Marwan, Dennis Bouvier, Cruz Izu, Natalie Kiesler, Teemu Lehtinen, Dominic Lohr, Andrew Petersen, and Sami Sarsa. 2022. Steps Learners Take When Solving Programming Tasks, and How Learning Environments (Should) Respond to Them. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2* (Dublin, Ireland) *(ITiCSE '22).* Association for Computing Machinery, New York, NY, USA, 570–571. https://doi.org/10.1145/3502717.3532168

[18] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2022. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems* 33, 2 (2022), 494–514. https://doi.org/10.1109/tnnls.2021.3070843

[19] Colin Johnson and Ursula Fuller. 2006. Is Bloom's taxonomy appropriate for computer science? *ACM International Conference Proceeding Series* 276 (02 2006), 120–123. https://doi.org/10.1145/1315803.1315825

[20] Andreas M. Kaplan. 2021. *Higher education at the crossroads of disruption: The university of the 21st century* (first edition ed.). Emerald Publishing Limited, Bingley, U.K. https://doi.org/10.1108/9781800715011

[21] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* 19, 1, Article 3 (2018), 43 pages. https://doi.org/10.1145/3231711

[22] Wendy Kicken, Saskia Brand-Gruwel, and Jeroen J.G. van Merriënboer. 2008. Scaffolding advice on task selection: a safe path toward self–directed learning in on–demand education. *Journal of Vocational Education & Training* 60, 3 (2008), 223–239. https://doi.org/10.1080/13636820802305561

[23] Maria Knobelsdorf. 2008. A typology of CS students' preconditions for learning. In *Proceedings of the 8th International Conference on Computing Education Research.* ACM, New York, NY and USA, 62–71.

[24] Michael Kohlhase. 2008. Using LaTeX as a Semantic Markup Format. *Mathematics in Computer Science* 2, 2 (2008), 279–304. https://doi.org/10.1007/s11786-008-0055-5

[25] Danny Kostons, Tamara van Gog, and Fred Paas. 2010. Self-assessment and task selection in learner-controlled instruction: Differences between effective and ineffective learners. *Computers & Education* 54, 4 (2010), 932–940. https://doi.org/10.1016/j.compedu.2009.09.025

[26] Dominic Lohr and Marc Berges. 2021. Towards Criteria for Valuable Automatic Feedback in Large Programming Classes. (2021).

[27] Uwe Maier, Thorsten Bohl, Marc Kleinknecht, and Kerstin Metz. 2013. Allgemeindidaktische Kategorien für die Analyse von Aufgaben. In *Lern- und Leistungsaufgaben im Unterricht*, Marc Kleinknecht (Ed.). Klinkhardt, Bad Heilbrunn.

[28] Melanie Margaritis, Johannes Magenheim, Peter Hubwieser, Marc Berges, Laura Ohrndorf, and Sigrid Schubert. 2015. Development of a competency model for computer science teachers at secondary school level. In *IEEE Global Engineering Education Conference*. IEEE Press, Los Alamitos, 211–220. https://doi.org/10.1109/EDUCON.2015.7095973

[29] Dennis Müller and Michael Kohlhase. 2022. sTeX3 – A LaTeX-based Ecosystem for Semantic/Active Mathematical Documents. In *TUGboat*. https://kwarc.info/people/dmueller/pubs/tug22.pdf to appear.

[30] Juliet Okpo. 2016. Adaptive Exercise Selection for an Intelligent Tutoring System. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, Julita Vassileva, James Blustein, Lora Aroyo, and Sidney D'Mello (Eds.). ACM, New York, NY, USA, 313–316. https://doi.org/10.1145/2930238.2930369

[31] Juliet A. Okpo, Judith Masthoff, and Matt Dennis. 2021. *Qualitative Evaluation of an Adaptive Exercise Selection Algorithm.* Association for Computing Machinery, New York, NY, USA, 167–174. https://doi.org/10.1145/3450614.3462240

[32] Florian Rabe and Michael Kohlhase. 2013. A Scalable Module System. *Information & Computation* 0, 230 (2013), 1–54. https://kwarc.info/frabe/Research/mmt.pdf

[33] Alexander Ruf, Marc Berges, and Peter Hubwieser. 2015. Classification of Programming Tasks According to Required Skills and Knowledge Representation. In *Informatics in Schools. Curricula, Competences, and Competitions*, Andrej Brodnik and Jan Vahrenhold (Eds.). Lecture Notes in Computer Science, Vol. 9378. Springer International Publishing, Cham, 57–68. https://doi.org/10.1007/978-3-319-25396-1{_}6

[34] Ron J.C.M. Salden, Fred Paas, and Jeroen J.G. van Merriënboer. 2006. Personalised adaptive task selection in air traffic control: Effects on training efficiency and transfer. *Learning and Instruction* 16, 4 (2006), 350–362. https://doi.org/10.1016/j.learninstruc.2006.07.007

[35] Katharina Schabram. 2007. *Lernaufgaben im Unterricht: instruktionspsychologische Analysen am Beispiel der Physik.* Ph. D. Dissertation. Duisburg-Essen, GER.

[36] Ana C. Stephens, Eric J. Knuth, Maria L. Blanton, Isil Isler, Angela Murphy Gardiner, and Tim Marum. 2013. Equation structure and the meaning of the equal sign: The impact of task selection in eliciting elementary students' understandings. *The Journal of Mathematical Behavior* 32, 2 (2013), 173–182. https://doi.org/10.1016/j.jmathb.2013.02.001

[37] John Sweller. 1988. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science* 12, 2 (1988), 257–285. https://doi.org/10.1207/s15516709cog1202{_}4

[38] Ottavia Trevisan, Marina de Rossi, and Valentina Grion. 2020. The positive in the tragic: Covid pandemic as an impetus for change in teaching and assessment in higher education. *Research on Education and Media* 12, 1 (2020), 69–76. https://doi.org/10.2478/rem-2020-0008

[39] Franz Emanuel Weinert. 2001. Concept of competence: A conceptual clarification. In *Defining and selecting key competencies*, Dominique Simone Rychen and Laura Hersh Salganik (Eds.). Hogrefe & Huber, Seattle, 45–65.

[40] Werner Wiater. 2011. *Unterrichtsplanung: Prüfungswissen - Basiswissen Schulpädagogik* (1. aufl. ed.). Auer, Donauwörth.

[41] Daniela Zehetmeier, Axel Böttcher, Anne Brüggemann-Klein, and Veronika Thurner. 2015. Development of a Classification Scheme for Errors Observed in the Process of Computer Programming Education. In *HEAd'15. Conference on Higher Education Advances.* Editorial Universitat Politècnica de València, 475–484.