Operationalizing the Continuum between Well-Defined and Ill-Defined Problems for Educational Technology

Nguyen-Thinh Le, Frank Loll, and Niels Pinkwart

Abstract—One of the most effective ways to learn is through problem solving. Recently, researchers have started to develop educational systems which are intended to support solving ill-defined problems. Most researchers agree that there is no sharp distinction but rather a continuum between well-definedness and ill-definedness. However, positioning a problem within this continuum is not always easy, which may lead to difficulties in choosing an appropriate educational technology approach. We propose a classification of the degree of ill-definedness of educational problems based on the existence of solution strategies, the implementation variability for each solution strategy, and the verifiability of solutions. The classification divides educational problems into five classes: 1) one single solution, 2) one solution strategy with different implementation variants, 3) a known number of typical solution strategies, 4) a great variety of solution strategies beyond the anticipation of a teacher where solution correctness can be verified automatically, and 5) problems whose solution correctness cannot be verified automatically. The benefits of this problem classification are twofold. First, it helps researchers choose or develop an appropriate modeling technique for educational systems. Second, it offers the learning technology community a communication means to talk about sorts of more or less ill-defined educational problems more precisely.

Index Terms—Ill-defined domains, ill-defined problems, ITS, CSCL, adaptive educational technology, classification

1 Introduction

One of the most effective ways to learn is through problem solving. Numerous directions for technology-enhanced learning have been devised, but just few are intended to support problem solving, including intelligent tutoring systems (ITS), computer-supported collaborative learning (CSCL), or computer-aided instruction (CAI). While ITSs primarily support students in solving problems through providing feedback, CSCL systems are intended to help students solve problems collaboratively using a computer system as a communication means. However, CSCL systems can also be enhanced with the capability of providing feedback to individuals or groups. In this paper, we refer to systems which can give feedback to student solutions for a given educational problem as *intelligent educational systems*. Researchers have successfully developed intelligent educational systems for well-defined problems whose solutions can be verified as correct or incorrect, for instance in algebra [1] or physics [2]. Recently, this research area has started considering ill-defined problems whose solutions are assessed subjectively (e.g., in categories such as usefulness, aesthetic quality, or elegance) [3]. In this paper, we focus on the nature of ill-definedness of educational problems and its impact on the design of appropriate learning technology.

• *The authors are with the Human-Centered Information Systems Research Group, Department of Informatics, Clausthal University of Technology, Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany. E-mail: {nguyen-thinh.le, niels.pinkwart}@tu-clausthal.de.*

Manuscript received 13 May 2012; revised 5 Dec. 2012; accepted 15 Mar. 2013; published online 10 Apr. 2013.
For information on obtaining reprints of this article, please send e-mail to: lt@computer.org, and reference IEEECS Log Number TLT-2012-05-0069.
Digital Object Identifier no. 10.1109/TLT.2013.16.

In the same domain, both well-defined and ill-defined problems can exist [4]. In the domain of programming, for example, human tutors may develop many types of programming problems. The following two assignments illustrate that:

1. Write a Java statement to sum the two numbers 4 and 5. Please fill in the missing operator:
   `S = 4 ___ 5;`
2. Develop an investment simulation system.

The first assignment can be considered a very well-defined problem, whereas the second one is very ill-defined, because it remains unclear what the simulation system should actually do and what the users would expect from it. We will elaborate on the ill-definedness of these example problems in the next section.

Most researchers agree that there is no sharp distinction between well-defined and ill-defined problems, but that there is a continuum between well-definedness and ill-definedness [5], [6], [7]. However, up to now a more specific classification or investigation of this "continuum" has not been proposed. Jonassen [8] attempted to represent a continuum between decontextualized problems with convergent solutions to very contextualized problems with multiple solutions using three types of problems: puzzle problems, well-defined problems, and ill-defined problems,[1] stating that these problem types do not represent a classification of well- or ill-definedness. Mitrovic and Weerasinghe [4] divided the space of problems into four quadrants based on ill-defined and well-defined instructional domains and tasks (the authors considered

1. Jonassen used the terms "well-structured" and "ill-structured" instead of "well-defined" and "ill-defined."

well-defined tasks within an ill-defined domain impossible so that one quadrant remained empty).

Is there a way to more specifically classify the *degree* of well- or ill-definedness of an educational problem, and what would be its value? While for some sorts of educational technology that do not provide intelligent support and adapt to the user, such as learning management systems, the classification would have little value, the situation is different for intelligent educational systems that need to interpret student solutions to give feedback. Here such a classification could be very helpful to help designers choose appropriate analysis and feedback provision techniques.

The classification of problems proposed in this paper is based on two dimensions. On a *qualitative* dimension, problems are classified based on the existence of alternative solution strategies, the implementation variability, and the solution verifiability. On a *quantitative dimension*, problems are distinguished based on the number of alternative solution strategies and the number of implementation variants. As a result, the classification consists of five classes of educational problems:

1. one single solution,
2. one solution strategy which can be implemented in different variants,
3. a known number of typical solution strategies which can be implemented in different ways,
4. a great variety of possible solution strategies (beyond the anticipation of human tutors) where each single solution can be assessed automatically as correct or not, and
5. problems whose solution correctness cannot be verified automatically.

In the next section, we review characteristics of ill-defined problems. Then, we illustrate two levels of solution variability for problems in three different example domains (geometry, travel planning, and programming): solution strategies and implementation variants. After that, we propose to classify educational problems into five classes according to the qualitative and quantitative criteria. Based on this classification, we analyze which educational technology approaches can be (or have been) applied to the different problem classes, and we argue why the classification is beneficial for educational technology researchers in at least two ways.

2 Ill-Defined Problems

In the literature, several different definitions for the terms "domain" and "problem" in the context of learning have been proposed [4], [5], [6], [7]. In this paper, we adopt the notions of "domain" and "problem" as suggested by Lynch et al. [7]. According to these authors, domains are considered as conceptual spaces or fields of study which can be represented in declarative or procedural knowledge. An educational problem within a domain has one or more goals that a solver must achieve. To solve an educational problem, the solver must apply relevant (declarative and procedural) knowledge to achieve the goals given initial knowledge. Thus, problems can be used to teach students about the domain. This paper focuses on problems rather than domains, following the argumentation of Fournier-Viger et al. [6] who suggested considering whether a problem is ill-defined and how it is ill-defined to choose appropriate domain knowledge modeling and reasoning techniques.

Several different definitions for the term "ill-definedness" have been proposed—indeed, one can argue that the term itself is somewhat ill-defined. In their review of these definitions, Lynch et al. [7] suggested that an ill-defined problem typically possesses the following characteristics:

1. the existence of open-textured concepts,
2. the lack of generally accepted domain theories,
3. a problem cannot be decomposed into independent subproblems,
4. the existence of prior cases which are facially inconsistent,
5. an analogical reasoning process using cases and examples is needed,
6. the existence of a large solution space,
7. the lack of formal methods to verify the correctness of a solution,
8. the lack of criteria to judge solutions,
9. the possibility to discuss solutions from different perspectives,
10. the disagreement among domain experts, and
11. the requirement to justify solutions.

The application of these criteria to the two programming problems introduced in Section 1 yields a fairly differentiated picture. For the first problem, regarding the first criterion for ill-definedness, no open textured concepts can be identified, because the problem statement is clearly formulated: A Java operator for an arithmetic calculation is required. The second criterion, the nonexistence of a formal theory, is also not satisfied by this programming problem. The underlying formal theory of programming is exactly the semantics of the machine model at hand and therefore directly available to all problems which address the formal aspects of programming. With respect to the third and sixth criterion, there is no possibility to "design" a solution and to divide the problem because the solution structure is already prespecified by an input slot, and thus, this programming problem does not fulfill these criteria. For such a programming problem, there exist no inconsistent prior cases because the syntax of a programming language defines clearly which addition operators can be used, and thus the fourth criterion is not fulfilled. The seventh and eight criterion are not fulfilled either, because the validity of the operator for an arithmetic calculation can be verified by the set of constructs of the programming language. In addition, to check whether the solution is correct with respect to the requirements of the programming problem, a set of test cases can be used. Since just a correct addition operator is required to solve this problem, no disagreement between domain experts may happen, and further discussion and justification can be spared, thus criteria 9, 10, and 11 are not satisfied. The fifth criterion seems to be relevant for this problem: the solver of this programming problem may compare some examples for arithmetic addition to reason about a correct addition

operator. However, this process is not required once the solver has learned the syntax of an addition operator or knows where to find the syntax reference manual for the programming language. We can conclude that the first programming problem is clearly well defined: all criteria of this definition suggest this.

For the second programming problem, the goal seems to be quite obvious. However, the existence of unspecified, open-textured concepts (criterion 1) is a phenomenon typically associated with complex programming problems. This is central and the source of other criteria of ill-definedness. Here, terminological clarifications and the development of a formal model are an integral part of finding a solution. For instance, the concept of a "net interest rate" is used to compute the return on investment for a given amount of money and will, thus, be relevant for the investment simulator. This concept, however, is defined differently in many financial institutions (e.g., depending on the service cost and whether the return is paid monthly, quarterly, or yearly), and thus needs to be specified. Furthermore, a special kind of open-textured concept can also be identified if metalevel aspects like efficiency, simplicity, elegance, or ease-of-use are considered. As a result, not only can solutions be considered from different perspectives (criterion 9), but even disagreement between domain experts concerning the appropriateness of solutions may occur (criterion 10), and the solver thus needs to justify her solution (criterion 11). With respect to verifying solutions, the validity of solutions cannot be verified easily, because no widely accepted formal theory can be applied. In the domain of investment, many theories exist, ranging from formal principles such as the return on investment to more informal ones such as a hedge. But no single theory can be proven correct (criteria 2, 7, and 8). Solving such a problem typically requires several interactions (discussion, negotiation, and agreements) with potential end users to clarify basic requirements of the problem and the solver can take prior cases or examples into account to reason about useful functions for an investment simulation systems (criteria 4 and 5). For example, the range of functions that the investment simulation system should offer may depend on individual preferences and conceptions—it is, therefore, open to debate. Similarly, evidence for ill-defined aspects can also be found with respect to criterion 6: an ill-defined problem has a large solution space. Obviously, programming is a constructive activity, at least at the level of complex programs such as an investment simulation system, much less of course at the level of "toy examples." Faced with a problem, a programmer usually has to choose between alternative ways of combining different programming constructs. Although the set of programming constructs is limited, these can be combined in many different ways to generate correct solutions. By looking at the internal interdependences of a problem solution, criterion 3 also provides evidence in favor of considering more ambitious programming problems as ill-defined. Decomposing a given problem into subproblems usually creates many dependences between the latter—for instance, a subroutine is usually used to solve a subproblem and the interface of the subroutine needs to be taken into account when using it. In summary, according to the criteria suggested in [7], the second problem can be considered ill-defined, whereas the first one is well defined.

Simon [5] considered the distinction between well-definedness and ill-definedness not as properties of a particular problem, but rather takes the perspective of the problem solving process which is characterized as a heuristic search procedure. To qualify as a well-defined problem, the solution process for the problem must have 1) uniquely specified start and end points, as well as 2) a formal procedure that describes the transition between the start and the end points, and 3) an evaluation function which verifies the correctness of the state transitions. Also according to these criteria, the first programming problem is well defined whereas the second one is not. In general, given a programming problem, the start point is the information given in the problem statement, and the end point is a program code that satisfies the requirements specified in the problem statement. To solve the first programming problem above, the formal procedure is described by the insertion of an appropriate operator for arithmetic calculation, and the evaluation function is defined based on the rules of the programming language being used. However, given a programming problem like the second one, a formal procedure cannot be specified easily because in addition to the task of applying the constructs of a programming language, the solver has to specify the requirements of an investment simulation system.

The two problems above illustrate the two ends of the continuum between well-defined problems and ill-defined problems in the same domain. Obviously, the solution space of the first problem (consisting of one solution) is smaller and simpler than that of the second problem (open-ended solution space). Can a more fine-grained insight into the structure of the solution space for a given problem be used to characterize the degree of ill-definedness of this problem? We will argue that this is indeed the case. Lynch et al. [7] pointed out that there is a causal relationship between the different characteristics of ill-defined problems. For example, due to the existence of open-textured concepts or the lack of generally accepted domain theories, many solution variants for a problem can be developed. Then, the developed solution variants need to be judged and discussed, where disagreements between experts can happen. In this light, we believe that the solution space to a problem is a key determining factor of ill-definedness, because one of the challenges of developing intelligent educational systems is analyzing solutions submitted by a student to provide appropriate feedback. Here, the space of solutions that the analysis needs to cover is an important aspect. It is connected to the characteristics of ill-definedness as stated by Simon in that this space becomes intractable if start or end state are undefined or ambiguous, and that the existence of (or lack of) transitions and evaluation functions have an impact on the options for automated analysis of elements (and their connections) within the space, and thus on the structure of the solution space.

3 Three Qualitative Characteristics of Solution Spaces

3.1 Alternative Solution Strategies

In the literature, the term "strategy" has been used to describe problem solving methods or algorithms. For example, Chi and VanLehn [9] identified forward chaining and backward chaining as two common problem-solving strategies in deductive domains.[2] In this respect, a "strategy" represents a general procedure to solve problems and is domain-independent. For example, the *target variable strategy* can be used to solve problems in many domains, including mathematics and physics [9].

By contrast, a *solution strategy* in our sense is dependent on domains and problems. It represents an approach or way to solve a specific given problem. Researchers have suggested that experts have knowledge about problem categories and associated *solution strategies* [10]. That is, when a problem is given to an expert, she will identify its characteristics by associating it with previously solved similar problems and apply the solution strategy which might typically be applied to solve problems of that type in the respective domain. A solution strategy differs from the notion of "solution schemas" [11] in that while a solution strategy represents just a general approach which can result in different solution structures, a solution schema represents the frame or a structure pattern of solutions. We exemplify the term *solution strategy* in the following three domains:

- In the domain of travel planning, a task could consist in finding a route between two places. Depending on available means of transportation, different strategies can be applied or even be combined: for instance, driving by car, taking a train, or taking a flight.
- In the domain of programming, alternative solution strategies are almost always available. If, for example, a task is to write a program to calculate the return on investment, a direct analytic computation or an iteration can be used. The latter solution strategy can be further refined into naive and tail recursion.
- In the domain of geometry, alternative solution strategies are based on the available theorems which have been proven. If a task is to prove that the triangle ADE is isosceles given that angle $\widehat{ABC}$ is equal to angle $\widehat{ACB}$ (see Fig. 1), we either have to prove that DE is parallel to BC to apply the isosceles triangle theorem, which states that the angles opposite the two equal sides of an isosceles triangle are equal, or apply the definition of isosceles triangles (i.e., show that $\mathrm{AD} = \mathrm{AE}$ holds).

Since feedback on solutions submitted by students is a means to help students improve their solutions, feedback should not be in conflict with the solution strategy a student is (apparently) applying. Otherwise, it could become useless or even confusing. This happens, for instance, if a student has planned a trip using the train, but an automated feedback message in the context of a car-based solution strategy is provided (note that using a car might well be part of a solution strategy which predominantly relies on railway connections).

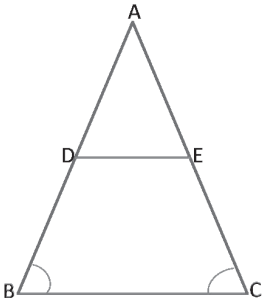

Fig. 1. A sample task in geometry.

In general, a solution strategy forms the basis for the process of finding a solution. However, it might be difficult to hypothesize the solution strategy a student applied from the student solution itself if the latter contains too little information about the solution process. For example, if a task is to find a correct number to replace the question mark in the equation: $12/15 = ?/5$, a typical wrong student answer is "2" from which a solution strategy can hardly be derived [12]. But if the solution to be provided by the student is richer in information (e.g., a travel plan, a proof, or a program), then there is a chance of inferring the solution strategy directly from the solution structure.

3.2 Implementation Variability

Once a problem solver has decided to use a specific solution strategy to solve a given problem, she is faced with the issue of how available means of the domain can be used for the implementation of that strategy. That includes finding out how to apply and arrange constructs of a particular domain in the context of the chosen solution strategy. We illustrate this in the three domains mentioned above.

- If in the domain of travel planning, a problem solver has chosen the strategy of using a car, she can find many different routes by combining different roads. Or, if traveling by train, the planner can also combine different train connections to reach the desired destination.
- In the domain of programming, a programmer has to apply the primitives of the programming language being used. Usually, there are many different options to implement a high-level concept. For instance, an arithmetic expression can be constructed using different combinations of arithmetic operators ($+, -, *, :, =, <, >$, etc.). Within an implementation, the sequential order of statements can sometimes be changed without changing the semantics. As a result, a solution strategy for a programming problem can be implemented in many ways.
- Similarly, if a solver has decided for a solution strategy to solve the geometry problem above (e.g., using the theorem that the triangle ADE is isosceles if DE is parallel to BC), there are multiple ways to arrange argument statements within the proof.

2. A task domain is deductive if solving a problem requires producing an argument, proof, or derivation consisting of one or more inference steps, and each step is the result of applying a domain principle, operator, or a rule.

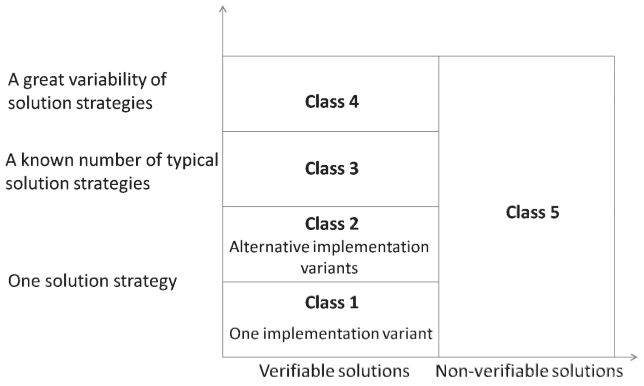

Fig. 2. Classification of educational problems.

3.3 Solution Verifiability

Some problems have solutions that cannot objectively be verified as correct or not. Rather, solutions of such problems can achieve a certain acceptance from a target group with respect to aspects such as aesthetics or usefulness. For those problems, the solution space becomes open-ended, because the acceptance of a solution can vary between individuals. The second programming problem (see Section 1) about the investment simulation system meets this attribute. Here, solutions cannot be verified as correct automatically.

4 Problem Classification

Using three qualitative attributes discussed in the previous section and also considering quantitative aspects (the number of alternative solution strategies and of implementation variants), we propose to classify educational problems into five classes (see Fig. 2). On the horizontal axis, the qualitative criterion "verifiability" distinguishes the first four classes of problems, which have verifiable solutions, from Class 5 problems whose solutions are nonverifiable. On the vertical axis, problems of Classes 1 to 4 are distinguished by the number of solution strategies and the variability of implementation options: one solution strategy (Classes 1 and 2), a known number of typical solution strategies (Class 3), and a great number of solution strategies (beyond the anticipation of a human tutor) (Class 4). These five problem classes represent an increasing degree of ill-definedness (from "very well-defined" to "highly ill-defined" (see Section 6 for a discussion of the sharpness of the classification).

Class 1: One solution strategy, one implementation. Problems of this class can be solved according to only a single solution strategy and have only one solution. They are specified in a way that the solution is unique, possibly enforced by a given solution structure or a multiple choice templates. Such problems are suited to recall basic knowledge of the domain being taught. Giving automated feedback to students for problems of this type is very easy: The system only needs to know the correct input. For instance:

- *How long is the* shortest route *for car driving from city A to city B?* ____.
- *Write a Java statement to sum two numbers 4 and 5. Please fill in the missing operator:* `S = 4 ____ 5;`
- *Given* $\widehat{ABC} = \widehat{ACB}$, *what can we say about the sides of the triangle?* `AB = ____.?`

Class 2: One solution strategy, alternative implementation variants. Problems of Class 2 can be solved according to a single solution strategy which, however, can be implemented in many different ways. Similar to the first level, problems on this level can typically be specified precisely so that the space of possible solutions is narrowed down to a single solution strategy, or the input is restricted by prespecified solution templates, for example:

- *Find a route for a car driving from city A to city B.*
- *Write a function to compute the return* $\boldsymbol{R}$ *of an investment* $\boldsymbol{X}$ *after* $\boldsymbol{N}$ *years for a fixed interest rate* $\boldsymbol{Y}$ *using a FOR-DO loop.*
- *Given* $\widehat{ABC} = \widehat{ACB}$, *prove that the triangle ADE is isosceles by applying the isosceles triangle theorem.*

To solve these problems, a student is not allowed to implement other solution strategies: The student is restricted to use cars, to choose the FOR-DO loop, or to apply the isosceles triangle theorem as the only single solution strategy. These restrictions narrow down the need for analysis of the student's intention (i.e., which solution strategy is pursued), and thus facilitate automated feedback provision. Problems of this class can be considered similar to "puzzle problems" [8] which have only one single correct solution, but several implementation variants for reaching this solution.

Class 3: A known number of typical solution strategies. For this class of educational problems, the student is free to choose among several known alternative solution strategies (which can be anticipated by a human tutor) and can implement the chosen solution strategy according to her preferences, for example:

- *Find a route from city A to city B (using one of several given available means of transportation).*
- *Write a function to compute the return on investment after a period for a fixed interest rate.*
- *Given* $\widehat{ABC} = \widehat{ACB}$, *prove that the triangle ADE is isosceles.*

This kind of problems is more challenging for students than problems of Classes 1 or 2, because they have to make appropriate design decisions between solution strategies and implementation variants while developing a solution, instead of simply applying a predefined solution strategy. This class of problems is not only challenging for students, but also for designers of intelligent educational systems. Feedback messages which are not in accordance with the student's solution strategy would be misleading—as such, either techniques to generate a reasonable hypothesis about the solution strategy pursued by the student correctly are required, or the system needs to be able to give feedback independent of solution strategies.

Class 4: A great variability of possible solution strategies while the correctness of any given specific solution can be verified automatically. Problems in this class are so complex that it may not be possible to a priori enumerate (and to encode into an intelligent educational system) all possible solution strategies that a student may pursue. The number of

possible solution strategies is beyond the anticipation of a human tutor. Also, if a problem needs to be solved by dividing it into (nonindependent) subproblems, where each of the latter can be solved using different solution strategies, the sheer number of subproblem solution combinations may be very large and unknown a priori if either the problem decomposition into subproblems or the number of solution strategies for any of the subproblem combinations is unknown. The following problems are examples for this class:

- *Develop a travel plan from city A to city B.* This problem can be solved by dividing a route into several parts and by combining different transportation means for parts of the travel route (including ones probably not anticipated by the task designer, such as "using a surfboard"). As a result, a great amount of combinations of possible routes is expected. Yet, any combination can be verified as correct if it leads from A to B.
- *Develop a calculator to calculate the return on investment.* To solve this task, a great number of design decisions have to be taken into account concerning many issues: the choice of appropriate data representations, the number and kind of input parameters considered, the presentation of the output, defining helper functions, and so on. Hence, the space of combinations of design decisions becomes large. However, the correctness of each solution can be verified using test cases.
- *Develop a formula to calculate the area of the triangle ADE.* In addition to the steps of applying possible solution strategies (based on available theorems) to prove the ADE is an isosceles triangle, the next task is to develop a formula to calculate the area of ADE. There are many ways to do this, including (but not restricted) to the following three, each of them derived from geometry theorems:
  - Applying the formula to calculate the area of a triangle directly: $\mathrm{a} = \frac{1}{2} * \mathrm{DE} * \mathrm{h}$, where h is the height of ADE.
  - Deriving the height of the isosceles triangle ADE from the Pythagorean theorem:

$$\mathrm{h} = \sqrt{AD^2 - \frac{1}{4} * DE^2}.$$

  - Applying trigonometry to derive the height of ADE: $\mathrm{h} = AD * cos(\frac{1}{2} * \theta)$, where θ is the vertex angle.

As a common point, for all problems of this class, even though there are many (possibly even unpredictably many) solution strategies, each solution could still be checked automatically (see McCarthy's definition of well-defined problems [13]). Providing appropriate feedback on an erroneous solution in accordance with the student's solution strategy, however, is more difficult than for solutions of Class 3 problems, since that would require an intelligent educational system to understand the solution strategy pursued by the student without having a list of "typical" strategies. That is the key distinction between problems of this class and Class 3. For instance, a classical chess problem is the following: given a chess board with some chess pieces, the goal of the solver is to achieve a task. For instance, the white player has to checkmate black in N moves. Given a known chess problem, an expert has a number of tactics (e.g., cross-check, decoy, deflection) and strategies (artificial castling, exchange) which can be considered as "solution strategies." Since these solution strategies can be anticipated, this chess problem is an instance of Class 3. However, if given a chess board with initial setting, a chess expert would not be able to estimate which solution strategy can be applied to checkmate the opponent within N moves (because the chess problem has to be redefined after a single move). As such, playing complete games of chess is an instance of Class 4. This is in consistence with the view of Simon [5] who suggested that chess playing is ill-defined when it is viewed as the play of an entire game, whereas it can be regarded as well defined if it is viewed as single moves.

Class 5: Multiple solution strategies, and solution correctness cannot be verified automatically. Solutions of problems of this class cannot be verified automatically. This can, for instance, occur if one criterion for a good solution is that it should be considered "useful" or acceptable by a large number of stakeholders. The latter requirement usually results in controversial opinions and renders solutions not formally verifiable. The following problems are instances of this class: *Develop the most relaxed travel plan from city A to city B; Develop an investment simulation system; Develop the most elegant formula to calculate the area of the triangle ADE.* Since the concepts "relaxed, elegant, investment system" are open-textured, they need to be interpreted and recharacterized. Actually, parts of these problems may be well defined, because some aspects of the problem solutions can be validated. For example, the task of developing a travel plan can be executed computationally (using an algorithm) and the resulting travel plans can be verified as correct or incorrect. Only the aspect "most relaxed" is ill-defined as discussed. This class of problems has also been named "wicked problems" [14] whose solutions are not right or wrong, but rather possible solutions need to be accepted by all stakeholders and thus are subject to debate.

Since an automated check for solution correctness is not possible for problems of this class, an automated provision of feedback on student solutions through intelligent educational systems is a nontrivial matter.

5 Problem Classification: The Benefits

The classification of educational problems into five classes with an increasing degree of ill-definedness, as proposed in the previous section, serves two purposes. First, it enables designers of intelligent educational systems to choose or develop an appropriate technique for dealing with educational problems (Section 5.1). Second, it provides the learning technologies research community a communication means to more precisely characterize sorts of educational problems (see Section 5.2).

5.1 Approaches for Building Intelligent Educational Systems

Researchers have devised numerous approaches to building intelligent educational systems. In the following, we review some of these approaches in the light of the proposed classification. Note that this discussion, while illustrating typical exemplary approaches suitable for the different classes, does neither claim to be exclusive nor exhaustive. Some approaches are suitable to be used across multiple classes (even though they are *typically* used for one class), and certainly the list of approaches discussed in this paper cannot be complete.

5.1.1 Computer-Aided Instruction for Class 1

Class 1 problems can be solved by applying a single solution strategy and have a single solution. Problems of this kind are usually referred to as drilling exercises and have often been deployed in CAI systems. These systems are intended to test whether students have acquired sufficient knowledge so far and to reinforce the required knowledge [15]. For example, AnimalWatch, a system which helps students solve arithmetic problems, poses the following problem [16, p. 23]:

> *A book says that one whale can eat 21 pounds of plankton in an hour. How many pounds can it eat it 7 hours? Enter your answer here:* _____

The task of the student is to input an appropriate number in the given solution template. The system checks the correctness of the student's solution. In the negative case, the system gives a hint indicating that the solution is incorrect without explaining the reason (because the answer provides too little information). In addition to indicating whether a student's answer is correct or not, CAIs are able to provide a correct answer. To check the correctness of solutions to such an exercise, the system simply needs to compare the student's solution against the prespecified value in the system's knowledge base.

5.1.2 Model-Based Approaches for Class 2

Problems of this level can be solved by applying a single solution strategy that can be implemented in many different variants. As compared to CAI systems, this allows addressing more challenging problems while still providing personalized feedback to students' solutions. Currently, model-tracing and constraint-based modeling (CBM) techniques are the two most prominent approaches which have been applied successfully for building ITSs for different domains. Model-tracing tutors have been built for many domains, including physics [2] and geometry [17]. In a model-tracing system, error diagnosis and instruction are carried out on the basis of an expert model and a set of buggy rules. This model represents one or more solution paths to a given problem (more if multiple implementation variants for the strategy exist). A solution path consists of many production rules, each of them composed as a pair of situation and action. Buggy rules represent typical erroneous problem solving paths of students. Whenever a student solution deviates from the expert model, the system identifies the situation where the student has carried out a wrong action.

While model-tracing systems are based on an expert model and buggy rules, a CBM tutor is built based on a predefined set of constraints. Researchers have successfully developed constraint-based ITSs for various domains, for instance, German grammar [18], or SQL [19]. Constraints can be used both to represent general knowledge of a domain [20] and to model properties of correct solutions to a specific given problem. Constraints of the former type are called principle constraints. Constraints for the latter purpose are referred to as semantic constraints and are used to check the semantic correctness of a student solution by comparing components of an ideal solution with the student solution [21]. While principle constraints are independent of problems (and, therefore, of solution strategies), semantic constraints are strategy-specific. Error diagnosis is based on evaluating constraints. If a solution violates a constraint, that means that the solution does not adhere to a principle of the domain being learned or does not satisfy a semantic requirement of correct solutions. The CBM approach is able to handle different implementation variants of the ideal solution, but it has not been designed with a specific focus on supporting multiple solution strategies. As a result, diagnostic information returned from a constraint-based tutor might mislead students if the solution strategy underlying the ideal solution is not the same as the strategy intended by the student [19], [22]. Therefore, the CBM approach (or, more specifically, the semantic constraints within this approach) is well suited for problems of Class 2, where only a single solution strategy is allowed, but problematic for higher classes of our classification hierarchy. Of course, principle constraints can still be used to model principled domain knowledge, and a CBM tutor can be built based on these constraints—but then, the feedback of this tutor is, of course, not specific to the problem at hand.

5.1.3 Strategy Recognizing Approaches for Class 3

Class 3 problems can be solved by applying several alternative solution strategies which can be implemented in many ways. In principle, the example approaches discussed in the previous subsection can be used to build intelligent educational systems for Class 3 as well. For instance, a CBM tutor without semantic constraints can be used for Class 3 (and higher) problems. Also, production rules can be used to model all possible correct solution paths for all possible implementation variants of multiple solution strategies. Andes, a model-tracing tutor for physics [2], is a representative of this approach. However, defining plausible expert models and buggy rules is a very laborious task even for one solution strategy, and even more for multiple strategies because one has to identify all possible ideal solution paths and enough buggy rules for each of these [22], [16, p. 85]. Except Andes, we are not aware of any other "classical" model-tracing tutor that is able to support alternative solution strategies. However, there are some other approaches—partly related to model-tracing or CBM—that are able to handle multiple solution strategies and still give problem-specific feedback to students.

Since creating the cognitive model for model-tracing tutors is laborious, Matsuda et al. [23] recently developed a machine learning technique called programming by

demonstration. In their approach, a "SimStudent" learns problem solving skills from humans and automatically generates production rules. The author of a model-tracing tutor, thus, simply needs to demonstrate possible solutions which may be instances of different solution strategies. The SimStudent generalizes those solutions and generates production rules for alternative solution strategies. This requires less effort than "classical" model-tracing tutor development, since demonstrating possible solutions for a given problem is less laborious than designing a cognitive model. The SimStudent worked very well: with a cognitive model generated based on 20 problems solved by a group of human students, the SimStudent was able to explain 82 percent of the problem-solving steps by another group of human students correctly [23].

Jeuring et al. [24] developed a tutoring system for the functional programming language Haskell, emphasizing the use of "programming strategies." For each problem contained in their system, alternative model solutions that represent different solution strategies are associated. Each solution strategy consists of a sequences of refinement rules (which refine programs) and rewriting rules which express a program construct in another way (this corresponds to the aspect of implementation variability in Section 3.2). For example, a problem can be split into two subproblems, solutions of which then together constitute solutions of the original problem. Using refinement rules of this (and other) type, errors in the student solution are detected and appropriate feedback is returned to the student. This approach is similar to model-tracing in that the refinement rules describe the routes leading to correct solutions. The authors claimed that the language which they use to describe solution strategies can be used in other domains in which procedures are expressed in terms of rewriting and refinement rules.

Le and Menzel [25] proposed to apply soft computing techniques to enhance the capability of error diagnosis for CBM tutors, because constraint-based error diagnosis is a constraint satisfaction problem (CSP) where the goal is to identify inconsistencies between an erroneous solution and a constraint system. Fargier and Lang [26] adopted a probabilistic approach for solving CSPs and developed a weighted constraint-based model (WCBM) for intelligent educational systems. In their approach, the process of diagnosing errors in a student solution consists of two interwoven tasks (hypotheses generation and hypotheses evaluation) which take place on two levels. First, on the *strategy level*, the system generates hypotheses about the student's intention by iteratively matching the student solution against multiple solution strategies specified in a semantic table. After each solution strategy has been matched, on the *implementation variant level*, the process generates hypotheses about the student's implementation variant by matching components of the student solution against corresponding components of the selected solution strategy. The generated hypotheses are evaluated with respect to their plausibility by aggregating the weight value of violated constraints. As a consequence, the most plausible solution strategy intended by the student is determined. Le and Pinkwart [27], [28] have shown that this approach is more effective in analyzing student's solution strategies correctly than a CBM tutor that has several sets of semantic constraints (one for each strategy) but does not use constraint weights.

For UML, Soler et al. [29] proposed a tool for teaching class diagrams. This tool assesses UML class diagrams provided by a student and gives feedback. For each problem, the system has a set of correct solutions (which represent different solution strategies). When a solution is entered by a student, the system compares it to the set of known correct solutions. The system then selects the correct solution which is most similar to the one proposed by the student and returns a corresponding feedback message to the student. For example, if the number of classes is incorrect, a message like "more/less classes are required" is returned. The system has been evaluated by comparing an experimental group and a control group. In the experimental group, the teacher used the system to perform some example exercises, then a personalized workbook with four exercises was assigned to each student. The control group used the same procedure and examples, but did not use the system. At the end, the students of both groups had to pass an exam, where the students of the experimental group outperformed their peers in the control group.

All the approaches discussed in this section have demonstrated to be appropriate for Class 3 problems. Whereas the SimStudent and the approach devised by Jeuring et al. [24] have been tested in several domains, the WCBM approach has been applied for the domain of logic programming only, and the approach of [29] is specific to UML. For problems of Classes 4 and 5, where the number of possible solution strategies is not a priori known (i.e., there may be strategies that are beyond the anticipation of a human tutor), the approaches may reach their limitation, because they all rely on a known number of typical solution strategies for each problem.

5.1.4 Educational Data Mining Techniques for Class 4

Intelligent educational systems for Class 4 problems, which are characterized by a great variety of solution strategies and by the fact that any solution is automatically verifiable, are rarely found in the literature. Most of the existing systems which support problems of this class apply data mining approaches. CanadarmTutor [30] can be considered as an educational system which has shown to support Class 4 problems. This system is intended to teach astronauts how to operate a robot manipulator deployed on the international space station (ISS). While solving the problem, the student does not have a direct view of the scene of operation on the ISS: the robot manipulation (i.e., moving the manipulator, berthing, mating) must rely on cameras mounted on the manipulator and at strategic places in the environment where it operates. Thus, it is not possible to define a complete domain model for such a robot manipulation problem. The problems provided by CanadarmTutor can be considered instances of Class 4, because given a robot manipulation problem, there are many possibilities for moving the robot to the goal position and each solution (a sequence of movements) can be verified as correct when the sequence of movements of the robot reaches a desired position. To deal with the issue of

the great variability of possible solutions of problems supported by CanadarmTutor, Nkambou et al. [31] proposed to combine two knowledge discovery techniques: sequential pattern mining and association rule mining. This approach assumes that actions of experts, intermediate, and novice users are recorded and annotated. Sequential pattern mining detects frequent action sequences of the annotated actions. Using the mined patterns, a generic basis of association rules, which represent links among the patterns, is generated. This aggregated information is then used to guide learners in problem-solving situations. The approach of combining different data mining techniques has demonstrated to be an effective way of building a domain model for an intelligent educational system automatically by mining knowledge from recorded structured/unstructured data.

Barnes and Stamper [32] used student data to create student models for an intelligent tutor for the domain of logic proof. Instead of modeling student behavior using production rules according to the model-tracing approach, the authors proposed to generate Markov decision processes (MDPs) representing possible student approaches to a particular problem. When a new student works on a problem, the student solution represented in graphs is matched against the MDPs. Using these MDPs, hints are generated automatically. This method reduces the intensive work for modeling expert's knowledge. Studies showed that this approach was able to generate hints over 80 percent of the time.

Recently, another data mining approach to building intelligent educational systems has been proposed. This approach can autonomously infer structures and feedback options from given data (e.g., student solutions) [33]. The proposed approach uses prototype-based learning methods and nonvectorial data structures, extended in a way that they allow to simultaneously structure solution spaces, learn metrics for structures, align student solutions with clusters of other solutions, and infer appropriate feedback based thereon. While the proposed approach has not been completely implemented, a first pilot validation of the approach was conducted using a data set from the domain of programming. The results show that clusters of structurally similar solutions could be detected, and that an automated provision of student feedback based on this clustering seems feasible.

5.1.5 Heuristic Techniques for Class 5

The most challenging problems for educational purposes, of course, are instances of Class 5. The main challenge here is that solutions cannot easily be judged right or wrong. This differentiates this class from the other classes. Nevertheless, it is often-times possible to distinguish between aspects of student solutions that are "typical" of good and poor solutions. A good solution is characterized by a high acceptance of end users, whereas a bad solution is not traceable by users that are familiar with the problem domain. Researchers have devised various approaches to building intelligent educational systems for this class of problems, including peer review, computer-supported collaborative argumentation, and case-based or rule-based reasoning techniques. As a common aspect, all these techniques rely on some form of heuristics that "guesses" if a student solution is good or not, and the system then provides feedback based on this estimation.

Peer review. One way of providing feedback on solutions of Class 5 problems is to rely on human judgment about the quality of student solutions. This can, for instance, be approached via peer review—an approach which is also used in other noneducational contexts, such as the quality management of scientific conferences and journals. In the learning context, this approach requests the system users to review and assess alternative solutions of their peers. This can be done in textual form or as a numeric assessment using scales. An example for the latter is the CITUC system [34] which has been used to support students in exam preparation. Here, the students assess the quality of solutions (for the same task) provided by their peers. Based on the aggregation of all assessments, a peer-review system is able to classify a student solution correctly. Alternative systems that make use of a textual peer-review approach include SWoRD [35] and PeerGrader [36].

In contrast to using only peer-reviews for the purpose of assessing student solutions (as the systems listed above do), Ogan et al. [37] proposed to define a multidimensional expert model using the results of an empirical analysis of relevant issues for building an intelligent educational system for the domain of intercultural competence. The expert model serves to rate the quality of student solutions based on both automated and human assessment. The human assessment is carried out by asking students to rate the solutions of their peers according to the dimensions specified in the expert model.

All the peer-review approaches have in common that the respective intelligent educational systems involve human knowledge to give feedback to students' solutions. A downside of this approach is that feedback often cannot be provided immediately, because humans themselves need time to analyze student solutions.

Computer-supported collaborative argumentation. While peer-review can be conceived as an indirect form of student interaction which enables the computer system (via the reviews) to assign a quality score to a student solution (in the numeric case), some systems favor a more direct way of student interaction to achieve a similar effect. A common practice to deal with problems of Class 5 is argumentation, where multiple users discuss about a problem pondering pros and cons until an agreement upon a solution that satisfies all discussants is found. The focus of automated support here typically is not on the content, but on the structure of argumentation. This argumentation process can be supported by computer tools providing (potentially shared) graphical representations like graphs, matrices or threads (see [38]). The elements available for this representation can be used to build constraints. An example is a language consisting of hypothesis and fact elements in combination with pro and contra relations. Here, the argumentation should follow a pattern typical for science: a hypothesis is stated and supported or falsified by facts. An intelligent educational system can be able to recognize weaknesses by means of these constraints—for instance, if a student does not specify a hypothesis at all, or if the

argument representation created by the student contains facts claimed to support each other (instead of supporting a hypothesis). To detect these kinds of weaknesses, various approaches have been proposed. The first approach, implemented in an early version of the Belvedere system [39] is pattern based. Here, students are only allowed to use predefined argumentation chunks to model their argument. Based on these argumentation chunks, the system is able to compare the student solution to an expert model of the same argument. However, this approach is work-intensive and not applicable to argumentation tasks where users are required to state their own opinions.

A second approach is the use of machine learning techniques. Based on experts' by-hand classification of argumentation moves, an intelligent educational system can learn to classify argumentation moves. This approach is used, for instance, in the ARGUNAUT system [40], which is an argumentation moderator assistant system in the sense that feedback will not be sent to the users who argue but to a human moderator. The reason for this is simple: the classification can result in erroneous diagnoses. The moderator's role is, therefore, to decide which kind of feedback is appropriate.

Another approach to detect weaknesses is using graph grammars as done in LARGO [41]. This system makes use of a set of predefined structural rules in graph grammar format that will be applied on demand to the overall argumentation structure. If a violation of one of the rules is detected, the system provides the student with a feedback message that highlights the possible weakness in the argument, and asks the student to self-explain this argument part (possibly changing it as a consequence).

The detection of structural weaknesses is only one possibility of analyzing student solutions. Another approach is checking the logical consistency, as for instance done by the ECHO algorithm of the argumentation system Convince Me [42]. Based on user assigned believability scores for each argumentation move, the system calculates the believability of the final conclusion and hence, the consistency of the overall argumentation. The user is asked to compare his final score with the one calculated by ECHO. Thus, the user is requested to check the internal logic of his argumentation.

Case-based and rule-based reasoning. Law students are often given study cases and have to find out how legal rules have been applied in past decisions and arguments [43]. Several researchers have adopted this strategy and have developed intelligent educational systems for the legal domain, including Thermis [44], Lites [45], or IKBALS II [46]. Case-based reasoning techniques can check the similarity between old cases to assess new problems. Rule-based reasoning techniques serve to model normative knowledge which enables the validation of a penal situation. Both techniques can be seen as heuristics: If cases are similar, then they should typically be decided in a similar way and thus the similarity can be exploited for intelligent educational systems—but this is, of course, not a rule without exceptions.

5.2 A Communication Means

The second benefit of the problem classification is that it serves the community of learning technology researchers and users as a communication means to characterize educational problems more precisely. In today's literature on learning technologies, we often find fuzzy descriptions about educational problems. In the domain of programming, for instance, we can find numerous vague characterizations for programming problems which are used to help students improve programming skills: "It [APT, a programming tutor] is designed as a programming environment to help students complete short programming assignments" [47, p. 152], "The ACT programming tutor helps students as they complete short programming exercises" [48], "Experiments have shown that, in the case of PROUST [a programming tutor for PASCAL], high performance in recognizing errors is achieved with simple programs. But the system has difficulties in understanding programs of a certain complexity and its capacity for identifying errors decreases drastically." [49, p. 131], "They [unit pages] include different interactive activities: simple questions and programming problems that the students can solve using the possibilities of WWW fill-out forms" [49, p. 135], "Java Intelligent Tutoring Systems (JITS) was designed to recognize small Java programs and provide intelligent feedback even when there is no authored solution available." [50, p. 50]. It remains unclear whether phrases such as *short programming exercises, simple programs, certain complexity, simple questions and programming problems, or small Java programs* have the same meaning and whether human tutors from different institutions would agree with these classifications: *simple programs* of a class in a K-12 school may be easier than *simple programs* provided in a course at the University graduate level. Therefore, we suggest using the problem classification to formulate these phrases more precisely and objectively, so that the capability of a computer-supported educational system can be described accurately.

Similar vague characterization of problems can also be found in the literature for other domains, for instance, in the domain of computational modeling using UML. Some existing intelligent educational systems which are intended to help students create class diagrams include Collect-UML [51], DesignFirst-ITS [52], and ACME-DB [29]. No doubt, creating a class diagram is a design activity and UML designs can be considered ill-defined. However, these systems provide different types of problems. Collect-UML supports students in creating UML class diagrams collaboratively. By adding a new class diagram component, "the students need to select the component's names from the problem text by highlighting or double-clicking on the words." [51, p. 167]. The system uses a prespecified *ideal solution* for each problem to compare to the student's class diagram. The authors of Collect-UML stated that "the system allows for alternative ways of solving a problem, as there are constraints that check for equivalent constructs between the student solution and the stored [ideal] solution" [51, p. 164]. However, due to the restriction of deciding for a new diagram element based on highlighted words or phrases in the problem description, this system narrows down the space of solution variability considerably. This certainly facilitates the analysis and can also be considered as beneficial from an educational point of view,

since the critical skill in UML modeling that students should focus on during learning is not the selection of names but the identification of types of components. Yet, this approach can also be critiqued. Moritz and Blank commented on Collect-UML as follows [52, p. 36]: "No free-form entry of element names is allowed. This approach avoids the need to understand the student's intent from natural language terms. A pedagogical drawback of this approach is that the student must eventually learn how to transfer to a less structured problem-solving environment." Since this system allows students to create different class diagram variants which are similar to a prespecified ideal solution, problems provided by this system can be classified as Class 2.

Different from Collect-UML, DesignFirst-ITS allows students to name components of a class diagram freely by adding classes, methods, and attributes. To diagnose errors in a student's class diagram, an Expert Evaluator module evaluates each step of the student's design process by comparing it with a *solution template* that has been created by an human instructor. To match the student's component name to a name in the solution template, the Expert Evaluator uses the SimMetrics algorithm to determine the similarity between the student's component name and component names in the solution template. Although the authors argued that "DesignFirst-ITS provides a relatively unstructured environment that allows students freedom to build their design." [52, p. 39], it still restricts students via the solution template specified by a human instructor. Thus, design problems supported by this system can be considered instances of Class 2.

Instead of checking the student's class diagram based on a single *ideal solution* or a *solution template*, ACME-DB compares the student's class diagram to a set of prespecified possible correct solutions for a design problem (see Section 5.1.3). Since this system supports different solution strategies represented by different possible correct solutions, problems supported by this system belong to Class 3 of the proposed problem classification. The authors of ACME-DB claimed that using this system, students are only restricted to attributes marked in brackets in the problem description. There is no restriction on names of classes or relationships. Classes are assessed by considering the set of attributes attached to them. Relationships are evaluated in terms of the classes they relate to.

As a conclusion, all three systems provide UML design problems to students to improve their computational modeling skills. The systems are very similar in nature. However, ACME-DB addresses Class 3 problems, while the others are designed to handle Class 2 problems.

6 Conclusions and Research Directions

In this paper, we have identified three qualitative attributes for characterizing problems for educational purposes: the existence of alternative solution strategies, the variability of implementation, and the verifiability of solutions. Based on these categories, a classification of problems has been proposed, ranging from "one solution strategy, one variant" to "a great number of solution strategies, and it's not possible to check a solution for correctness." This classification can be conceived as a scale for the degree of well- or ill-definedness of a problem. The more possible implementation variants and the more possible solution strategies are possible, the more ill-defined is a problem. The nonverifiability of solutions determines the highest degree of ill-definedness in our classification. The very well-defined problems are least complex and can be solved by submitting a correct value (i.e., Class 1 problems). The most ill-defined problems are most complex because in addition to the existence of many alternative solution strategies, solutions for these kinds of problems cannot be verified as correct or not correct automatically (Class 5).

Based on this classification, we have discussed example approaches for building intelligent educational systems for each class of problems (see [4], [7] for other review and example approaches). While "classic" CAI methods are only applicable within Class 1, today's most prominent ITS paradigms (model-tracing and CBM) are typically used for Class 2, where only one single solution strategy is allowed. Some approaches have been proposed to extend the range of applications for CBM and model-tracing to also Class 3 problems, which are characterized by a known number of (a priori known) solution strategies, so that the system has to determine the strategy pursued by the student. However, these approaches are not applicable to problems of Class 4 where the number of available solution strategies may be very large (or even unknown). Intelligent educational systems for Class 4 problems are indeed rare as of today—yet, as has been argued, approaches with data mining components have the potential to be used for problems of this class. For Class 5 problems, where it is not possible to determine if a solution is correct or not, we have identified a variety of approaches relying on heuristic techniques such as peer review, collaborative computer-supported argumentation, case-based reasoning, and rule-based reasoning. The problem classification and this review are intended to help designers of intelligent educational systems choose a corresponding modeling approach for a class of problems. In addition, the problem classification serves as a communication means to characterize educational problems more precisely.

We are aware that the borderline between Classes 3 and 4 is not sharp. One can argue that for both classes, a number of possible solution strategies is possible. The difference between these two problem classes is the number of possible solution strategies which can be anticipated by human tutors. We suggest that a problem whose solution strategies can be anticipated by a human tutor should be considered an instance of Class 3. Otherwise, if for a problem there exist a range of many possible solution strategies which are beyond the anticipation of a human tutor (where the correctness of any given specific solution can be verified automatically), this problem is an instance of Class 4. This difference, even if not sharp, is of enormous relevance for educational technology designers. For Class 3, they can build complete models that cover all solution strategies, while this is not feasible for Class 4.

The classification does not say anything about sizes of solution spaces. It may well be the case that the solution space of a problem of Class 2 is larger than one of a Class 3 problem. For instance, a problem of Class 2 may have only

one solution strategy but many implementation variants, whereas another problem of Class 3 may have only few solution strategies with few implementation variants. Also, the classification does explicitly *not* target at judging or comparing the quality or suitability of intelligent educational system approaches. It certainly is not the case that an approach which is usable for Class 5 problems is "better" than one that can only be used for Class 1 problems. In fact, one can even argue that approaches (like CBM or model-tracing) typically used for Classes 1 or 2 have proven highly effective in that they can give specific, helpful support to students—while many approaches for the "higher" classes struggle with possibly erroneous diagnoses and less specific feedback options.

A surprising result of the classification is that, while systems for problem Classes 1 and 2 are very well known and have been investigated over decades, also for Class 5 there are quite a few systems which take different approaches for "guessing" the correctness of a student solution (and the solution strategy that the student may have pursued). Yet, for the intermediary Classes 3 and 4, which are arguably simpler than Class 5, only relatively few approaches have been proposed, and empirical evidence of their success is sparse. Of course, approaches that can handle Class 5 problems are also applicable to Classes 3 and 4 problems, but such a use does not fully exploit the potential for educational technology construction (e.g., error-prone heuristics or costly peer reviews would not really be required for Classes 3 and 4). Also, Classes 3 and 4 are really important for education, since problems of these classes allow for multiple solution strategies and can thus encourage critical, creative thinking, and decision-making.

The classification and the review of approaches also showed some opportunities for future research. For instance, educational data mining techniques have demonstrated their potential as an effective approach for Class 4 problems. In general, as one way of constructing Class 4 intelligent educational systems, one could imagine a combination of methods for Classes 1-3 and a collaborative data collection approach. Based on the collected data, alternative good and poor solution strategies to a problem could be learned and could be used to generate feedback even for Class 4 problems. To our knowledge, there is currently no approach which is able to do so.

For problems of Class 5, building a complete domain model for diagnosing errors and student's intention automatically may be an infeasible task. Therefore, researchers currently try to combine successful modeling techniques (e.g., model-tracing and CBM) with other educational approaches (collaborative argumentation, peer-review). Although this approach cannot return feedback to the student's solution immediately (because humans need time for analyzing solutions), this kind of combination seems to be a fruitful way to support students solve problems of this level and should be investigated further.

There exist numerous approaches for developing intelligent educational systems, and many of them have demonstrated their success for educational problems of Classes 1 and 2, where solutions are limited to a single solution strategy. Here, not much choice is left to students as they construct solutions. While this may be OK (and perhaps even desirable) for introductory lessons and problems to help the student focus on the skills to be learned, it considerably narrows down the possibilities of applying (or even critically comparing) different solution strategies and thereby experiment with domain constructs. In future research, intelligent educational systems should be able to provide problems of different classes to students adaptively. That is, after students have mastered problems of Classes 1 and 2, they should have the opportunity to face more ill-defined problems (e.g., problems of Classes 3-5).

REFERENCES

[1] K.R. Koedinger, J.R. Anderson, W.H. Hadley, and M.A. Mark, "Intelligent Tutoring Goes to School in the Big City," *Int'l J. Artificial Intelligence in Education,* vol. 8, pp. 30-43, 1997.

[2] K. VanLehn, C. Lynch, K. Schulze, J.A. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill, "The Andes Physics Tutoring System: Lessons Learned," *Int'l J. Artificial Intelligence in Education,* vol. 15, no. 3, pp. 147-204, 2005.

[3] C. Lynch, K. Ashley, A. Mitrovic, V. Dimitrova, N. Pinkwart, and V. Aleven, *Proc. 10th Int'l Workshop on Intelligent Tutoring Systems and Ill-Defined Domains Held at the 10th Int'l Conf. Intelligent Tutoring Systems (ITS '10),* 2010.

[4] A. Mitrovic and A. Weerasinghe, "Revisiting Ill-Definedness and the Consequences for ITSs," *Proc. 14th Int'l Conf. Artificial Intelligence in Education (AIED '09),* pp. 375-382, 2009.

[5] H.A. Simon, "The Structure of Ill Structured Problems," *Artificial Intelligence,* vol. 4, no. 3, pp. 181-201, 1973.

[6] P. Fournier-Viger, R. Nkambou, and E. Nguifo, "Building Intelligent Tutoring Systems for Ill-Defined Domains," *Proc. Advances in Intelligent Tutoring Systems Conf.,* pp. 81-101, 2010.

[7] C. Lynch, K.D. Ashley, N. Pinkwart, and V. Aleven, "Concepts, Structures, and Goals: Redefining Ill-Definedness," *Int'l J. Artificial Intelligence in Education,* vol. 19, no. 3, pp. 253-266, 2009.

[8] D.H. Jonassen, "Instructional Design Models for Well-Structured and Ill-Structured Problem-Solving Learning Outcomes," *Educational Technology Research and Development,* vol. 45, no. 1, pp. 65-94, 1997.

[9] M. Chi and K. VanLehn, "Meta-Cognitive Strategy Instruction in Intelligent Tutoring Systems: How, When, and Why," *Educational Technology and Soc.,* vol. 13, pp. 25-39, 2010.

[10] J.-M. Hoc, *Cognitive Psychology of Planning.* Academic Press, 1988.

[11] M.L. Gick, "Problem-Solving Strategies," *Education Psychologist,* vol. 21, pp. 99-120, 1986.

[12] S. Ohlsson and N. Bee, "Radical Strategy Variability: A Challenge to Models of Procedural Learning," *Proc. Int'l Conf. Learning Science,* pp. 351-356, 1991.

[13] J. McCarthy, "The Inversion of Functions Defined by Turing Machines," *Automata Studies,* 1956.

[14] J. Conklin, "Wicked Problems & Social Complexity," *Dialogue Mapping: Building Shared Understanding of Wicked Problems,* Wiley, 2005.

[15] P. Suppes, M. Jerman, and D. Brian, *Computer Assisted Instruction: Stanford's 1965-66 Arithmetic Program.* Academic Press, 1968.

[16] B.P. Woolf, *Building Intelligent Interactive Tutors.* Morgan Kaufman, 2009.

[17] V. Aleven and K.R. Koedinger, "An Effective Metacognitive Strategy: Learning by Doing and Explaining with a Computer-Based Cognitive Tutor," *Cognitive Science,* vol. 26, no. 2, pp. 147-179, 2002.

[18] W. Menzel, "Diagnosing Grammatical Faults—A Deep-Modelled Approach," *Proc. Third Int'l Conf. AI: Methodology, Systems, Applications,* pp. 319-326, 1988.

[19] B. Martin, "Intelligent Tutoring Systems: The Practical Implementation of Constraint-Based Modelling," PhD dissertation, Univ. of Canterbury, 2001.

[20] S. Ohlsson and E. Rees, "The Function of Conceptual Understanding in the Learning of Arithmetic Procedures," *J. Cognition and Instruction,* vol. 8, no. 2, pp. 103-179, 1991.

[21] S. Ohlsson and A. Mitrovic, "Constraint-Based Knowledge Representation for Individualized Instruction," *Computer Science and Information Systems,* vol. 3, no. 1, pp. 1-22, 2006.

[22] V. Kodaganallur, R. Weitz, and D. Rosenthal, "An Assessment of Constraint-Based Tutors: A Response to Mitrovic and Ohlsson's critique of 'a Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms,'" *Int'l J. Artificial Intelligence in Education,* vol. 16, pp. 291-321, 2006.
[23] N. Matsuda, W.W. Cohen, J. Sewall, G. Lacerda, and K.R. Koedinger, "Evaluating a Simulated Student Using Real Students Data for Training and Testing," *Proc. 11th Int'l Conf. User Modeling,* pp. 107-116, 2007.
[24] J. Jeuring, A. Gerdes, and B. Heeren, "A Programming Tutor for Haskell," *Proc. Central European School on Functional Programming Conf.,* pp. 1-45, 2011.
[25] N.-T. Le and W. Menzel, "Using Weighted Constraints to Diagnose Errors in Logic Programming—The Case of an Ill-Defined Domain," *Int'l J. Artificial Intelligence in Education,* vol. 19, no. 4, pp. 381-400, 2009.
[26] H. Fargier and J. Lang, "Uncertainty in Constraint Satisfaction Problems: A Probabilistic Approach," *Symbolic and Quantitative Approaches to Reasoning and Uncertainty,* vol. 747, pp. 97-104, Springer, 1993.
[27] N.T. Le and N. Pinkwart, "Adding Weights to Constraints in Intelligent Tutoring Systems: Does It Improve the Error Diagnosis?" *Proc. Sixth European Conf. Technology Enhanced Learning,* pp. 233-247, 2011.
[28] N.-T. Le and N. Pinkwart, "Can Soft Computing Techniques Enhance the Error Diagnosis Accuracy for Intelligent Tutors?" *Proc. 11th Int'l Conf. Intelligent Tutoring Systems,* pp. 320-329, 2012.
[29] J. Soler, I. Boada, F. Prados, J. Poch, and R. Fabregat, "A Web-Based E-Learning Tool for UML Class Diagrams," *Proc. IEEE Education Eng. Conf.,* pp. 973-979, 2010.
[30] P. Fournier-Viger, R. Nkambou, and E.M. Nguifo, "Exploiting Partial Problem Spaces Learned from Users' Interactions to Provide Key Tutoring Services in Procedural and Ill-Defined Domains," *Proc. Conf. Artificial Intelligence in Education,* pp. 383-390, 2009.
[31] R. Nkambou, P. Fournier-Viger, and E.M. Nguifo, "Learning Task Models in Ill-Defined Domain Using an Hybrid Knowledge Discovery Framework," *Knowledge-Based Systems,* vol. 24, no. 1, pp. 176-185, 2010.
[32] T. Barnes and J.C. Stamper, "Automatic Hint Generation for Logic Proof Tutoring Using Historical Data," *Educational Technology and Soc.,* vol. 13, no. 1, pp. 3-12, 2010.
[33] S. Gross, X. Zhu, B. Hammer, and N. Pinkwart, "Cluster Based Feedback Provision Strategies in Intelligent Tutoring Systems," *Proc. 11th Int'l Conf. Intelligent Tutoring Systems,* pp. 699-700, 2012.
[34] F. Loll and N. Pinkwart, "Using Collaborative Filtering Algorithms as Elearning Tools," *Proc. 42nd Hawaii Int'l Conf. System Sciences,* pp. 1-10, 2009.
[35] K. Cho and C.D. Schunn, "Scaffolded Writing and Rewriting in the Discipline: A Web-Based Reciprocal Peer Review System," *Computers and Education,* vol. 48, no. 3, pp. 409-426, 2007.
[36] E. Gehringer, "Electronic Peer Review and Peer Grading in Computer-Science Courses," *Proc. 32nd Technical Symp. Computer Science Education,* pp. 139-143, 2001.
[37] A. Ogan, V. Aleven, and C. Jones, "Advancing Development of Intercultural Competence through Supporting Predictions in Narrative Video," *Int'l J. Artificial Intelligence in Education,* vol. 19, pp. 267-288, 2009.
[38] O. Scheuer, F. Loll, B.M. McLaren, and N. Pinkwart, "Computer-Supported Argumentation: A Review of the State-of-the-Art," *Int'l J. Computer-Supported Collaborative Learning,* vol. 5, no. 1, pp. 43-102, 2010.
[39] D.D. Suthers, "Representational Guidance for Collaborative Inquiry," *Arguing to Learn,* Computer-Support Collaborative Learning Series, J. Andriessen, M. Baker, and D.D. Suthers, eds. vol. 1, pp. 27-46, Springer, 2003.
[40] R. de Groot, R. Drachman, R. Hever, B.B. Schwarz, U. Hoppe, A. Harrer, M. de Laat, R. Wegerif, B.M. McLaren, and B. Baurens, "Computer Supported Moderation of E-Discussions: The ARGUNAUT Approach," *Proc. Conf. Computer-Supported Collaborative Learning,* pp. 168-170, 2007.
[41] N. Pinkwart, K. Ashley, C. Lynch, and V. Aleven, "Graph Grammars: An ITS Technology for Diagram Representations," *Proc. 21st Int'l FLAIRS Conf.,* pp. 433-438, 2008.
[42] P. Schank and M. Ranney, "Improved Reasoning with Convince Me," *Proc. Conf. Companion on Human Factors in Computing Systems (CHI '95),* pp. 276-277, 1995.
[43] V. Aleven, K.D. Ashley, and C. Lynch, "Helping Law Students to Understand US Supreme Court Oral Arguments: A Planned Experiment," *Proc. 10th Int'l Conf. AI and Law,* pp. 55-59, 2005.
[44] I. Bittencourt, E. Costa, B. Fonseca, G. Maia, and I. Calado, "Themis, a Legal Agent-Based ITS," *Proc. 13th Int'l Conf. Artificial Intelligence in Education Workshop,* pp. 11-20, 2007.
[45] G. Span, "LITES, an Intelligent Tutoring System for Legal Problem Solving in the Domain of Dutch Civil Law," *Proc. Fourth Int'l Conf. AI and Law,* pp. 76-81, 1993.
[46] G. Vossos, J. Zeleznikow, T. Dillon, and V. Vossos, "An Example of Integrating Legal Case Based Reasoning with Object-Oriented Rule-Based Systems: IKBALS II," *Proc. Third Int'l Conf. AI and Law,* pp. 31-41, 1991.
[47] F.P. Deek and J. McHugh, "A Survey and Critical Review of Tools for Learning Programming," *J. Computer Science Education,* vol. 8, no. 2, pp. 130-178, 1999.
[48] A. Corbett and J.R. Anderson, "Student Modeling in an Intelligent Programming Tutor," *Cognitive Models and Intelligent Environments for Learning Programming,* E. Lemut, B. du Boulay, and G. Dettori, eds., pp. 1-10, Springer, 1993.
[49] M. Gomez-Albarran, "The Teaching and Learning of Programming: A Survey of Supporting Software Tools," *Computer J.,* vol. 48, no. 2, pp. 130-144, 2005.
[50] E.R. Sykes, "Qualitative Evaluation of the Java Intelligent Tutoring System," *J. Systemics, Cybernetics and Informatics,* vol. 3, no. 5, pp. 49-60, 2005.
[51] N. Baghaei, A. Mitrovic, and W. Irwin, "Supporting Collaborative Learning and Problem-Solving in a Constraint-Based CSCL Environment for UML Class Diagrams," *Int'l J. Computer-Supported Collaborative Learning,* vol. 2, no. 2, pp. 159-190, 2007.
[52] S. Moritz and G. Blank, "Generating and Evaluating Object-Oriented Designs for Instructors and Novice Students," *Proc. Nineth Int'l Conf. Intelligent Tutoring Systems Workshop Intelligent Tutoring Systems for Ill-Defined Domains,* pp. 35-45, 2008.

Nguyen-Thinh Le received the PhD degree in intelligent educational technology systems for programming from the University of Hamburg, Germany. He is a lecturer and researcher in the Department of Informatics, Clausthal University of Technology. Currently, he performs research in crowd-sourcing and data mining for educational technology systems.

Frank Loll received the PhD degree in domain-independent support for computer-based education of argumentation skills from Clausthal University of Technology in 2012.

Niels Pinkwart is a professor for the Human-Centered Information Systems Reasearch Group at Clausthal University of Technology. He has a research background at the intersection of computer science, educational and collaborative technology, and human-computer interaction. His research interests include adaptive educational technologies for ill-defined problems.