Towards a Classification for Programming Exercises

Nguyen-Thinh Le and Niels Pinkwart

Humboldt Universität zu Berlin
Germany

{nguyen-thinh.le, niels.pinkwart}@hu-berlin.de

Abstract. When researchers of the AIEDCS (AI-supported Education for Computer Science) community want to exchange programming exercises as baselines for e.g., evaluation purposes, several questions related to the difficulty of exercises will arise: What kind of programming exercises are supported by an intelligent learning environment? How difficult are the programming exercises? In this paper, we investigate programming exercises supported by fifteen existing intelligent learning environments for the domain of programming and have learned that these exercises can be classified into three classes: 1) exercises with one single solution, 2) exercises with different implementation variants, and 3) exercises with different solution strategies. The contribution of this classification is two-fold. First, it can be used to help designers of intelligent learning environments for programming apply/devise appropriate modeling techniques for a specific class of programming exercises that are intended to support the programming/learning process of students. Second, it helps researchers of the AIEDCS community communicate more accurately when they want to discuss programming exercises.

Keywords: Programming exercises, classification, intelligent tutoring systems

1 Introduction

Usually, in a programming course of an undergraduate or graduate program, students are requested to solve programming exercises using a language's interpreter (if the machine model of the language requires), a compiler, and a text editor or an integrated development environment. That is, students have to solve programming problems by developing programs in free-form.

A variety of intelligent learning environments for programming has been devised and employed in order to help students acquire problem solving skills and principles in the domain of programming by providing, for instance, adaptive feedback or Socratic dialogues in natural language as a communication means. In order to apply or devise appropriate AI techniques, the initial question is how programming exercises should look like. It is very challenging for building an intelligent learning environment where students are allowed to solve programming exercises in free-form as they usually do, because powerful capability of understanding student's solutions needs to

be modeled in the system. Thus, when developing an intelligent learning environment for programming, it is required to make a trade-off between restricting the problem solving creativity of students and developing a powerful intelligent technique in order to understand a large space of possible solutions submitted by students. Depending on the type of programming exercises, appropriate AI techniques have been devised for building intelligent learning environments for programming. If researchers intend to exchange programming exercises for, e.g., evaluation purposes, the first question they would ask is: Is the difficulty level of programming exercises to be exchanged appropriate?

The aim of this paper is to identify the common characteristics of programming exercises supported by existing intelligent learning environments for programming and proposing a classification for programming exercises. The goal is to use this classification for programming exercises to define a shared task of collecting and sharing programming exercises (and students' solutions) between researchers of the AIEDCS community.

2 Related Work

With respect to learning outcomes, there exist several educational learning taxonomies among that Bloom's taxonomy [4] and SOLO taxonomy [3] are most widely used. These taxonomies can be used to design courses at various levels of granularity (e.g., [10]), to design teaching and assessment materials (e.g., [14]), and to analyze student responses to exercises in order to measure student's progress (e.g., [21]).

In the context of categorizing educational problems that are used to develop problem solving skills, Le et al. [12] proposed a general (i.e., not domain specific) classification based on three qualitative dimensions: 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) problems that have one single solution, 2) problems that can be solved by applying one solution strategy with different implementation variants, 3) problems that can be solved with a known number of typical solution strategies, 4) problems that have 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. This classification has been developed for categorizing educational problems in a general sense. Here, in this paper, we propose to apply it in the context of programming exercises.

3 A Classification for Programming Exercises

Before programming exercises can be classified, we investigate typical exercises that have been supported by existing intelligent learning environment for programming. For that purpose, we have selected fifteen systems that were used in quantitative and/or qualitative evaluation studies (Table 1). The first column of the table

shows the name and literature of the system. The second column describes the programming language to be learned and level of the target group. The third column reproduces a sample of typical programming exercises supported by a specific system. In order to keep the description of the exercises authentic, we will cite them literally. Some sentences in bold depict the interface given to the student to solve a programming exercise. The interface is usually illustrated by a picture in literature.

Table 1. Typical programming exercises supported by intelligent learning systems

System	Language & Level	Exercises
Lisp-Tutor [2]	LISP; No information about level oft he target group is available	„Define **rightp**. It takes three arguments **side1**, **side2**, and **side3**, which represent the lengths of of the three sides of a given triangle, where **side3** is the longest side. **rightp** tests whether the three sides form the right triangle. In a right triangle, the sum of the squares of **side1** and **side2** is equal to the square of **side3**. However, your function should allow for a measurement error of 2 percent. In other words, your function should return **t** if the sum of the squares of the first two sides is within 2 percent of the square of the third side. For example **(rightp 3 4 5.01) = t**, because 3 squared + 4 squared = 25, which is within 2 percent of 5.01 squared." **A structured editor which automatically balances parentheses is given. When the student types a LISP keyword, a new template is presented.**
JITS [16, 17]	Basic Java constructs (variables, operators, looping structures); First programming course at the College and University level	"Problem: Write a program called "Exponentiation" which calculates 2^N, where N is a user specified number. For example, if N were assigned the value 4, then the result would be 2^4 = 2x2x2x2 = 16. Program specifications: This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program. OUTPUT>Result: 16" **A skeleton program is given.**
ELM-ART [20]	LISP; Introductory university course	<u>Exercise Type 1:</u> "Define a function CUBOID-VOLUME-NEW. This function expects as its argument a three elements containing the side lengths of the cuboid. Example: (CUBOID-VOLUME-NEW * (2 4 5).). 40 (CUBOID-VOLUME-NEW * (10 50 6).). 3000 (CUBOID-VOLUME-NEW * (0 1000 2).). 0

		The self-defined function MY-SECOND and MY-THIRD can be used in the construction of the function. Type in your solution here:" **A text-box for the answer is given.**
		Exercise Type 2: "What is the result of evaluating the expression? (ERROR if the evaluation results in an error): (REST '(A B C D))" **A text-box for the answer is given.**
Hong [9]	Prolog (no information about level of the intended target group)	"Write a program which reverses the elements of a list into another list." (**If the student requests help, then a template is given**) "reverse(<arguments>). reverse(<arguments>):- <pre-redicates>, reverse(<arguments>),post-predicate>."
Kumar [11]	C++; CS2	**A program is given.** "Assume that all the parameters are passed by Value-Result when main() calls the function convert(). Indicate the final value of the variable area and all the changes to the elements of the array depth after the execution of the program." **Slots for entering values are given.**
Ludwig [15]	C++; Introductory University level	The system "allows the student to edit his or her programs in a controlled text editor, offer programs for analysis, then submit them for grading."
Truong [18]	Java; "beginning students" of Information Technology	"Write a simple program that obtains two integer values – lowerLimit and upperLimit from the user. Display all integers between owerLimit and upperLimit in ascending order." **An interface with slots is given.**
Al-Imamy [1]	C; Principles of Business Programming course	"The user will have the program's main structure and the control (X) to delete any statement (except the main structure) and the control (U) to add a new statement. Pressing the control (U) will display a list of all valid statements at the location." **A program template with slots is given.**
JTutors [6]	Java APIs; Introductory course	"a quiz allows the student to test his knowledge of the API by filling the blanks"
HabiPro [19]	Java: Programming course at the university level	4 types of exercise: Exercise Type 1: finding the mistake in a program "Correct the mistake in this exercise" **A program is given.** **A text-box for the answer is given.**
		Exercise Type 2: put a program in the correct order "To solve the exercises the screen is divided into two parts. In the first one the disarranged program appears. When the students choose a sentence it is automatically put into the

		second window. At the end, students check if they have ordered the program correctly. In this case, the program is presented with comments and indenteds."
		Exercise Type 3: predicting the results "At the beginning, programs without comments and variables with random names will be shown to the students and they must guess what the program does. Next, a similar program will be shown but in this case the program will have adequate comments and significant names of variables so that students can see that in the second case it is easier to trace a program"
		Exercise Type 4: completing a program "Students must write one sentence that is omitted. In this exercise we try to make sure that there are different solutions. The system only accepts the best solution."
JavaGuide [8]	Java; Introductory programming classes	Parameterized questions "What is the final value of result?"
QuizPack [5]	C; Programming-related classes	Parameterized questions
Ask-elle [7]	Haskell; Bachelor students at the university level	"Write a function that creates a list with all integers between a given range: range :: Int ! Int ! [Int] For example: > range 4 9 [4, 5, 6, 7, 8, 9]" **The name of the function along with its parameters is displayed:** range x y = *
INCOM [13]	Prolog; Graduate University level	"Calculate the return after investing an amount of money at a constant yearly interest rate" **A program template with two slots is given: one slot for clause head, another slot for clause body.**

Adopting the three qualitative dimensions that Le et al. [12] employed for classifying educational problems, we analyze programming exercises that have been supported by existing intelligent learning environments. We attempt to assign them into appropriate classes.

Class 1: Programming exercises have a single correct solution. Programming exercises of this class are usually given in form of a quiz which consists of a program, a question and a gap to be filled with correct value. When solving such programming exercises, students have to understand the given program code and input a correct solution into a pre-specified gap. Using such programming environments, we cannot trace problem solving steps of students. The following systems provide programming exercises of this class: JavaGuide [8], QuizPack [5], Exercise Type 2 provided by

ELM-ART [20], JTutors [6]. HabiPro [19] also supports exercises of this class (Exercise Type 1 and Type 3), for them one single correct solution is expected. Another type of programming exercises also belongs to this class: these are characterized through a template consisting of several slots in addition to an exercise description (e.g., Kumar's system [11], Truong's system [18], Al-Imamy's system [1]). These systems allow students to input only specific correct values into slots. If one slot is filled with an incorrect value, then the student will receive feedback accordingly.

Class 2: Programming exercises can be solved by different implementation variants. Programming exercises of this class are provided with a specific exercise description. In addition to a problem statement, usually, a specification about the solution strategy to be applied is given or a program skeleton is pre-specified. Since students have the possibility to modify values (i.e., program statements), alternative implementation variants can be developed. The following systems support programming exercises of this class: JITS [16, 17]; ELM-ART [20]; Hong's system [9] in guiding mode where it provides a program template that contains several slots, and thus, many implementation variants are possible; HabiPro's exercises of Type 2 [19] allow students to change the order of program's statements and for this type of exercises, alternative implementation variants are possible.

Class 3: Programming exercises can be solved by applying alternative solution strategies. Programming exercises of this class enable students to apply different solution strategies which have been anticipated by the exercise's author(s) and allow students to implement each solution strategy in different variants. The following systems provide programming exercises of this class: the LISP-tutor is able to trace different problem-solving steps of a student and disambiguate the intended solution strategy of the student by asking him/her to select one of available helper functions from a menu; In the automatic error analysis mode, Hong's system [9] supports problem solving in free-form; HabiPro's exercises of Type 4 allow students to apply different solution strategies; Ask-elle [7] gives hint in addition to an exercise description that the student can choose one of proposed solution strategies and for each solution strategy, the student can develop alternative implementation variants by refining the holes (denoted by *); Using INCOM [13], students have the option to add many Prolog clauses and subgoals as required, thus, alternative solutions strategies are possible and since the slots (that represent a clause head and a clause body) can be filled with code in different ways, alternative implementation variants can be developed.

From the investigation of fifteen systems above, we could not identify programming exercises that correspond to Class 4 proposed in [12]. This can be explained by the fact that domain knowledge modeled in these systems is limited by the number of solution strategies that can be anticipated by the exercise's author. E.g, Ask-elle requires that students implement one of pre-specified solution strategies; INCOM compares the similarity between the student's solution and the solution strategies (that are represented in form of a semantic table) and gives feedback in the context of the most plausible anticipated solution strategy; Hong's system and HabiPro have domain models that contain a set of reference solutions which represent alternative solution strategies. Programming exercises of Class 5 [12] are not supported by existing intel-

ligent learning systems in literature, neither. Solutions of programming exercises of this class cannot be verified automatically, because in addition to the formal correctness of a program, more subjective criteria (e.g., aesthetics, low consumption of computing resources) need to be considered.

4 Relations between the Classification for Programming Exercises and PISA-Levels of Proficiency for Mathematics

The previous section showed that, in the classification of [12], the existing programming learning environments support classes 1 through 3. How does this compare to other, more general, classifications? One prominent international programme that provides such classifications is PISA (Programme for International Student Assessment). While PISA has not proposed a specific literacy model for Computer Science, frameworks for Mathematics, Reading, Science, Problem Solving and Financial literacy[1] have been published. As there are considerable similarities between Mathematics and Computer Science literacy, we next compare the proposed classification for programming exercises and the PISA proficiency levels in Mathematics. PISA defines six levels of proficiency in Mathematics (cf. Appendix) whereas the highest level (level six) demands highest cognitive capability, and thus this proficiency scale represents an empirical measure of the cognitive demand for each question/exercise.

The classification of programming exercises proposed in this paper suggests that Class 1 consists of exercises that require students to fill a correct value in a gap or to fill several values into several pre-specified slots. In order to be able to input correct value(s) into a gap or slots, the student need to understand a given program or a quiz. That is, they need to understand given information and to be able to process them. This requirement is in accordance with the description of proficiency Level 1 that requires that "they [students] are able to identify information and to carry out routine procedures…They [students] can perform actions that are obvious…"

Class 2 of programming exercises consists of exercises that provide a specific exercise description. Usually, a specific solution strategy to be applied is given. In order to solve exercises of this class correctly, students not only need to understand information given in the exercise, but also they need to be able to apply a specific solution strategy that is described in the exercise statement. That is, they need to establish the relevance of the suggested solution strategy with a specific problem situation. In addition, they need to be able to implement the specific solution strategy correctly. This required capability is in line with the requirements of proficiency Level 2: "Students can interpret and recognize situations in contexts that require no more than direct inference. They can extract relevant information from a single source and make use of a single representational mode. Students at this level can employ basic algorithms, formulae, procedures, or conventions".

[1] PISA 2012 Frameworks - Mathematics, Problem Solving and Financial Literacy
http://www.oecd.org/pisa/pisaproducts/pisa2012draftframeworks-mathematicsproblemsolvingandfinancialliteracy.htm

Class 3 of programming exercises allows students to apply different solution strategies anticipated by the human tutors. In order to be able to solve programming exercises of this class, students need to know in advance what kind of solution strategies are appropriate to solve the given programming problem, then they are required to implement one of possible solution strategies correctly. Thus, this requirement corresponds to the proficiency Level 3 that "students can execute clearly described procedures, including those that require sequential decisions. They can select and apply simple problem solving strategies." In addition to the capability of choosing an appropriate solution strategy and implementing a solution strategy correctly, in order to solve programming exercises of this class, students also need to be able to explain their problem solving steps (e.g., HabiPro), and this capability is required by proficiency Level 4 that requires that "they [students] can construct and communicate explanations and arguments based on their interpretations, arguments, and actions."

We have argumentatively attempted to establish a connection between the classification for programming exercises and the proficiency levels for Mathematics. Of course, this needs to be validated empirically using representative programming exercises of each class. We can notice that no programming exercises supported by existing intelligent learning environments for programming match proficiency Level 5 and Level 6 for Mathematics.

5 Conclusion

In this paper, we have applied the classification for education problems provided in [12] in order to classify programming exercises supported by existing intelligent learning environments. Based on investigation of fifteen existing intelligent learning environments for programming, three classes of programming exercises have been identified: 1) exercises with one single solution, 2) exercises with different implementation variants, and 3) exercises with different solution strategies. The contribution of this classification is twofold. First, designers of intelligent learning environments can choose or develop an appropriate modeling technique for a specific class of programming exercises. Second, this classification serves researchers of the AIEDCS community with a communication means to talk about sorts of programming exercises more precisely when they intend to exchange their learning materials (programming exercises), e.g. for evaluation purposes. We compared the proposed classification for programming exercises with the proficiency scale for Mathematics of PISA. We could identify correspondence between the Class 1 and Class 2 of the classification for programming exercises with proficiency Level 1 and Level 2, whereas cognitive demands for Class 3 programming exercises are in line with proficiency Level 3 and Level 4, and no classes of programming exercises that have been reviewed fulfill requirements of proficiency Level 5 and Level 6.

6 References

1. Al-Imamy, S.; Alizadeh, J. & Nour, M. A.: On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. Journal of Information Technology Education, vol. 5, 271-283 (2006).
2. Anderson, J. A. and Reiser, B. J.: The LISP tutor: it approaches the effectiveness of a human tutor. Journal Byte, 10(4), 1985, pp. 159-175.
3. Biggs, J.B. and Collis, K.F.: Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome). Academic Press, New York (1982).
4. Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H. and Krathwohl, D.R.: Taxonomy of Educational Objectives: Handbook 1 Cognitive Domain. Longmans, London (1956).
5. Brusilovsky, P. and Sosnovsky, S.: Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK. Journal on Educational Resources in Computing (JERIC), ACM, 5 (2005).
6. Dahotre, A.: jTutors: A Web-Based Tutoring System For Java APIs (2011). http://hdl.handle.net/1957/20562
7. Gerdes, A.; Jeuring, J., and Heeren, B.: An interactive functional programming tutor. In Proceedings of the 17th ACM annual conference on Innovation and Technology in Computer Science Education, 250-255 (2012).
8. Hsiao, I.-H.; Sosnovsky, S. A., and Brusilovsky, P.: Adaptive Navigation Support for Parameterized Questions in Object-Oriented Programming. In Proceedings of the 4th European Conference on Technology Enhanced Learning EC-TEL, Springer, 88-98 (2009).
9. Hong, J.: Guided programming and automated error analysis in an intelligent Prolog tutor. International Journal of Human-Computer Studies, Academic Press, 61, 505-534, (2004).
10. Howard, Richard A., Carver, Curtis A. and Lane, William D.: Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: tying it all together in the CS2 course. Proceedings of the 27th SIGCSE Technical Symposium on CS education, ACM (1996).
11. Kumar, A. N.: Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. Technology, Instruction, Cognition and Learning (TICL) Journal, 4 (2006).
12. Le, N. T., Loll, F. and Pinkwart, N.: Operationalizing the Continuum between Well-defined and Ill-defined Problems for Educational Technology. IEEE Journal Transactions on Learning Technologies, 6(3), 258-270 (2013).
13. Le, N. T. and Pinkwart, N.: Adding Weights to Constraints in Intelligent Tutoring Systems: Does it Improve the Error Diagnosis? In Proceedings of the 6th European Conference On Technology Enhanced Learning (ECTEL), 233 – 247, Springer Verlag (2011).
14. Lister, R.: On Blooming First Year Programming, and its Blooming Assessment. Proceedings of the Australasian Conference on Computing Edu., ACM Press, 158-162 (2000).
15. Shaffer, S. C.: Ludwig: an online programming tutoring and assessment system. The SIGCSE Bulletin, ACM, 37(2), 56-60 (2005).
16. Sykes, E. R.: Qualitative Evaluation of the Java Intelligent Tutoring System. Journal of Systemics, Cybernetics and Informatics, 3(5), 49-60 (2006).
17. Sykes, E. R. and Franek, F.: Inside the Java Intelligent Tutoring System Prototype: Parsing Student Code Submissions with Intent Recognition. In Proceedings of IASTED International Conference Web-based Education (2004).
18. Truong, N., Roe, P. and Bancroft, P.: Static analysis of students' Java programs Proceedings of the Sixth Australasian Conference on Computing Education – Vol. 30, Australian Computer Society, Inc., 317-325 (2004).

19. Vizcaíno, A., Contreras, J., Favela, J., and Prieto, M.: An Adaptive, Collaborative Environment to Develop Good Habits in Programming. In Proceedings of the 5th International Conference on Intelligent Tutoring Systems, Springer-Verlag, 262-271 (2000).
20. Weber, G. and Brusilovsky, P.: ELM-ART: An Adaptive Versatile System for Web-based Instruction. International Journal of AI in Education, 12, 351-384 (2001).
21. Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. K.A., and Prasad, C.: An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies. In Proceedings of the 8th Australasian Conference on Computing Edu., vol. 52, Australian Computer Society, Inc, 243-252 (2006).

Appendix: PISA 2012: Six levels of Mathematics Proficiency

"At Level 6 students can conceptualise, generalise, and utilise information based on their investigations and modelling of complex problem situations. They can link different information sources and representations and flexibly translate among them. Students at this level are capable of advanced mathematical thinking and reasoning. These students can apply this insight and understandings along with a mastery of symbolic and formal mathematical operations and relationships to develop new approaches and strategies for attacking novel situations. Student at this level can formulate and precisely communicate their actions and reflections regarding their findings, interpretations, arguments, and the appropriateness of these to the original situations.

At Level 5 students can develop and work with models for complex situations, identifying constraints and specifying assumptions. They can select, compare, and evaluate appropriate problem solving strategies for dealing with complex problems related to these models. Students at this level can work strategically using broad, well-developed thinking and reasoning skills, appropriate linked representations, symbolic and formal characterisations, and insight pertaining to these situations. They can reflect on their actions and formulate and communicate their interpretations and reasoning.

At Level 4 students can work effectively with explicit models for complex concrete situations that may involve constraints or call for making assumptions. They can select and integrate different representations, including symbolic, linking them directly to aspects of real-world situations. Students at this level can utilise well-developed skills and reason flexibly, with some insight, in these contexts. They can construct and communicate explanations and arguments based on their interpretations, arguments, and actions.

At Level 3 students can execute clearly described procedures, including those that require sequential decisions. They can select and apply simple problem solving strategies. Students at this level can interpret and use representations based on different information sources and reason directly from them. They can develop short communications reporting their interpretations, results and reasoning.

At Level 2 students can interpret and recognise situations in contexts that require no more than direct inference. They can extract relevant information from a single source and make use of a single representational mode. Students at this level can employ basic algorithms, formulae, procedures, or conventions. They are capable of direct reasoning and making literal interpretations of the results.

At Level 1 students can answer questions involving familiar contexts where all relevant information is present and the questions are clearly defined. They are able to identify information and to carry out routine procedures according to direct instructions in explicit situations. They can perform actions that are obvious and follow immediately from the given stimuli."