Evaluation of a UML-Based Versus an IEC 61131-3-Based Software Engineering Approach for Teaching PLC Programming

Birgit Vogel-Heuser, *Senior Member, IEEE*, Martin Obermeier, Steven Braun, Kerstin Sommer, Fabian Jobst, and Karin Schweizer

***Abstract*—A field experiment investigated the evaluation, teaching, and application of two different approaches to automatic control in programmable logic controllers, in particular comparing the Unified Modeling Language (UML) to the classic procedural paradigm (IEC 61131-3). A total of 85 apprentices from a vocational school for production engineering with a specialization in mechatronics took part in the training and the experiment. This paper details the results of the training using both approaches, and the correlations found between the modeling and/or programming performance and cognitive abilities, interest, workload, expertise, and school grades. In general, the results show that students can be trained to carry out authentic programming tasks within one and a half days, even for beginners in programming. The data distinguish the two approaches. Function Block Diagram programming (IEC 61131-3) can be best predicted by the grade in mathematics, programming experience, and cognitive demand. For performance in UML class diagram and state chart $(\mathrm{UML/CD} + \mathrm{SC})$ modeling, the grade in mathematics plays an even more prominent role; this explains the greater variance in modeling performance in the UML group than in the 61131/Function Block Diagram group. With respect to other findings, the paper concludes that special problem-solving skills and skills for abstract thinking should be taught when teaching UML-based modeling approaches.**

***Index Terms*—Automation, cognitive science, engineering education, modeling, object-oriented methods, programmable logic devices.**

I. Introduction

MODEL-DRIVEN object-oriented (OO) software engineering of programmable logic controllers (PLCs) remains a challenge in machine and plant automation [1]. Since the IEC 61131-3 standard [2] is widely accepted in industrial automation [3], using OO features for model construction in this field is still uncommon [4]. According to Thramboulidis and Frey [3], the benefit of integrating IEC 61131 with Unified Modeling Language (UML) lies in being able to use several diagrams in order to capture more aspects of the system. This gives a more complex yet still comprehensible model. Furthermore, it is assumed that the use of more diagrams leads to the generation of more abstract mental models.

Several works attempt to integrate IEC 61131 with UML or SysML, e.g., [5]–[10]. However, previous research on software engineering with UML shows that while using more diagrams is confusing [10], using adapted subsets of UML is helpful [7], [8], [11].

This paper presents a study comparing the teaching of classic procedural PLC programming versus teaching object-oriented PLC programming based on UML. Control courses teaching PLC programming or programmable logic design are mandatory for electrical, mechanical, and computer engineering undergraduates in Germany and elsewhere (cf. [12, Table I] and [13]). In the Faculty of Mechanical Engineering of the Technische Universität München (TUM), Munich, Germany, PLC programming in IEC 61131-3 and UML is integrated into lectures on automation and the associated automation lab course offered in the fifth semester of the Mechatronics and Mechanical Engineering Bachelor's degree programs; this course is mandatory for the Mechatronics students. Further courses for vocational and technical schools have been developed from these lectures.

The research question posed was whether the chosen approach to teaching beginners in programming and OO modeling was successful in facilitating modeling. A further question was whether correlations could be found between a student's performance in modeling and other variables (e.g., [14]–[17]) that could explain their difficulties in learning OO modeling. Correlations were therefore sought between performance with either the model-based OO approach using UML [class diagram and state chart $(\mathrm{CD} + \mathrm{SC})$], or the classical PLC programming approach using IEC 61131-3/FBD, and students' cognitive abilities, interest, workload, expertise, and school grades in various subjects. The main hypotheses were that the correlations would differ for each approach (modeling versus programming), and that UML modeling is more demanding and requires certain cognitive abilities, namely problem solving. When using UML, mentally representing and defining suitable states and operators gradually minimizes the discrepancy between the initial and the end state [18]. Section II briefly discusses the state of the art in software engineering for production automation to introduce the two approaches. Section III discusses teaching requirements and the programmers' knowledge of automation. Section IV describes the design of the experimental study and presents the results. Finally, Section V discusses the results and future work.

Manuscript received July 17, 2012; revised September 21, 2012; accepted October 10, 2012. Date of publication November 29, 2012; date of current version July 31, 2013. This work was supported in part by the Universität Bayern e.V. and the Vocational School for Manufacturing Technology Munich and their trainees, who participated in the experiment.

B. Vogel-Heuser, M. Obermeier, S. Braun, and K. Sommer are with the Institute for Automation and Information Systems, Technische Universität München, 85748 Garching, Germany (e-mail: vogel-heuser@ais.mw.tum.de; obermeier@ais.mw.tum.de; braun@ais.mw.tum.de; sommer@ais.mw.tum.de).

F. Jobst and K. Schweizer are with the University of Education Weingarten, 88250 Weingarten, Germany (e-mail: jobst@ph-weingarten.de; schweizer@ph-weingarten.de).

Digital Object Identifier 10.1109/TE.2012.2226035

II. State of the Art in Software Engineering for Production Automation

The IEC61131-3 standard defines five procedural programming languages for PLC programming [2]: two textual programming languages, Instruction List (IL) and Structured Text (ST), and three graphical languages, Function Block Diagram (FBD), Ladder Diagram (LD), and Sequential Function Chart (SFC).

Lucas and Tilbury [19] investigated different metrics, taking one sample of code each from LDs, Petri nets, and modular finite state machines to evaluate the effectiveness of various logic control design methodologies using task analysis. According to [20], the time and manpower required either to create a program, to install and debug a program on a machine, or to change an existing program can be regarded as alternative measures. In this context, Venkatesh *et al.* [21] suggest counting the number of elements required to represent a certain program to measure its complexity; Lee and Hsu [22] translated programs into Boolean expressions and counted the number of these required.

This paper focuses primarily on comparing the teaching of automation software engineering using either OO modeling with UML in [23]–[25] or procedural programming with IEC 61131-3/FBD. UML specifies different diagram types that set different priorities. In the following, the focus is on class and state chart diagrams as important chart types for the preparation of OO modeling.

Subsequently, the UML CD as a structure-focused type of diagram will be discussed in detail. CDs are designed to represent the system structure in the form of classes. A class is defined as a summary of objects of similar properties and functionality. It contains attributes and methods that apply to all elements associated with the class. The level of abstraction of a class is dependent on the respective task. Associations and generalizations between classes are linked to the structure of a programming task. A software structure can be realized by associating classes with other classes, as well as by creating generalizations between them. Generalizations specify inheritance from general classes to more specialized ones.

In contrast to the CD, an SC diagram describes a program from a behavioral perspective. It shows all discrete states of an object during runtime. Basically, an SC contains two components: states where methods of classes are invoked, and transitions with conditions leading to the next state. The UML-Plugin for CoDeSys 3, a reference implementation of the OO extension of IEC 61131-3 [8], provides modeling, coding, and online debugging of UML models, where OO elements can be mixed with traditional IEC language elements. Supported diagram types are CDs for structural description, and SCs and activity diagrams for behavioral description. The online debugging functionality is entirely integrated into the IEC 61131-3 environment, thus the program/model can be monitored during runtime [7].

III. Teaching Requirements and Programmers' Knowledge

Most studies investigating the benefit of OO programming fail to show clear results, perhaps due to different affordances in teaching. Thramboulidis [23], [24] used constructivist methods when teaching OO programming and received good evaluations from his students. Another approach, which is pursued here, is to examine the competencies related to different programming approaches.

A different type of authentic (and therefore also constructivist) teaching was chosen by Kim and Jeon [26], who addressed the problem of how to teach freshmen the basic concepts of embedded systems and how these could be applied to real-world problems. They used LEGO Mindstorms to empower students to build their own LEGO robots and ANSI-C to program and operate them; students scored an average of 56% correct answers.

Berges and Hubwieser investigated Computer Science freshmen's abilities to learn OO programming in two and a half days with as little (human) instruction as possible [27]. Examining 300 students' program code, they found that most were able to write quite satisfying programs. They identified two types of students: those who accept and apply the OO concepts, and those who prefer to program in a more traditional procedural way. They also tried to define the characteristics of object orientation to evaluate measures for program quality, e.g., one instance of a class is created.

The problem how to teach freshmen, novices, and students of introductory courses in computer science or engineering is also addressed by [26] and [28]–[34]. Faux [28] investigated whether a preprogramming course designed as a breadth-first approach affected students' ability to perform coding tasks. He found that students evaluated the sections on problem solving, algorithm development, and pseudocode as being the most important. Jacobson *et al.* [29] also regarded problem-solving skills as a prerequisite of design skills for engineering students. Among other measures, the authors assessed students' abilities by a short multiple-choice questionnaire on problem solving, theoretical mathematics and physics, English, and the use of SI units. In order to better develop design skills, hands-on measurements and communication were stressed. Verginis *et al.* [30] employed a Web-based, adaptive, activity-oriented learning environment (SCALE) in a blended learning setting and trained 175 students for eight weeks. One of the main conclusions was that successful students learned from the feedback from the system, especially when they had initially submitted wrong answers. Like [27], they discovered two types of students: those who tried to guess the right answer rapidly (27.8%), and those who responded correctly after considering the system's tutorial feedback or other relevant educational material (72.2%).

Boticki *et al.* [33] analyze the educational benefits of introducing the aspect-oriented programming paradigm, in addition to the object-oriented programming paradigm, into a programming course for undergraduate Software Engineering students. How the additional aspect-oriented paradigm affects students' programs, their exam results, and their overall perception of the theoretical benefits of aspect-oriented programming was assessed by automated analysis of student-created computer programs, surveys, and exam results. Their results show that "the use of aspect-oriented programming as a supplement to object-oriented programming enhances the productivity of novice program code software engineering students and leads to increased understanding of theoretical concepts" [33].

Further papers on programmers' competencies in modeling and informatics systems application are related to competence models, e.g., [14], [15], and [35]. Competence models often refer to curricula and syllabi. Usually, competencies in this context are understood as abilities, skills, and knowledge—a perspective that is still prominent in most Anglo-American research on competencies. An example is provided by Curtis [16], who proposed that programming results depend on individual personal factors and mental abilities. His model covers intellectual aptitudes, the knowledge base, cognitive styles, motivational structure, personality characteristics, and behavioral characteristics. Although Curtis did not empirically test his model, the factors would appear to have validity.

Other approaches for gaining insights into the competencies required for different programming approaches or skills are to analyze interviews from experts in the relevant domain [36] or to evaluate programmers' behavior when performing certain tasks, such as programming [32] or debugging [35]. This type of research often uses Bloom's taxonomy, e.g., [37], dividing cognitive aspects of learning into six hierarchical levels (recall of facts, comprehension, application, analysis, synthesis, and evaluation). Applying that to different programming tasks, Lahtinen [31] identified six types of student behavior in a cluster analysis of novice software engineering students: competent, practical, unprepared, theoretical, memorizing, and indifferent.

Kim and Lerch [38] realized that repetition is required to learn object orientation. This point of view is also emphasized by Ruocco [32] and the studies reported above, i.e., [26]–[31]. He selected a phased approach to threading UML throughout a Computer Science degree program and found that incorporating UML in the database course and incorporating use case diagrams as well as sequence and activity diagrams generated a richer and deeper exposure to UML.

In summary, three facts are important for the present study.

1) Teaching computer science or object orientation to beginners or freshmen requires repeated training, as shown by longitudinal studies [26]–[32].
2) Certain types of curricular organizations that emphasize problem solving and other cognitive abilities favor the successful teaching of programming and modeling, e.g., [7], [9], and [11].
3) It is important to investigate aspects of cognitive learning on different levels, e.g., [32], [35], and [39].

IV. Experimental Study

The aim of this empirical study was to investigate whether the chosen approach to teaching beginners in OO modeling was successful, and whether correlations could be defined between students' modeling performance and other variables, which could explain their difficulties in learning OO modeling. The performance measures varied with the Bloom taxonomy levels for cognitive aspects of learning [39]. Patig [40], [41] and Gemino and Wand [42] present results of various experiments on the usability of modeling notations. In order to establish fair conditions for informationally equivalent software engineering approaches, different training scenarios and experimental problem-solving situations were created, and a suitable task setting and training, supported by Hierarchical Task Analysis (HTA), was developed [4], [41]. The complexity of the training and the test was increased, based on Kim [38], but "isomorphic" problems were used. The characteristics of object orientation were defined to evaluate measures for the program quality based on Berges and Hubwieser [27], e.g., by creating one instance of a class.

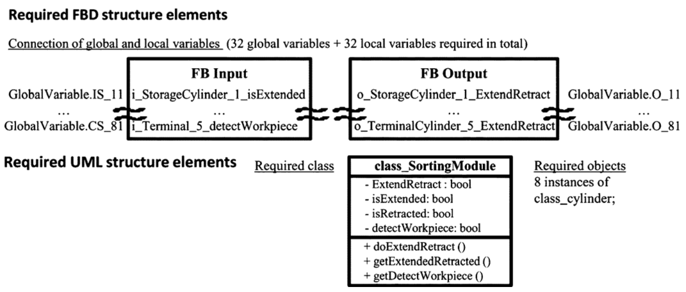
Required FBD structure elements
Connection of global and local variables (32 global variables + 32 local variables required in total)
FB Input
GlobalVariable.IS_11
i_StorageCylinder_1_isExtended
...
GlobalVariable.CS_81
i_Terminal_5_detectWorkpiece
FB Output
o_StorageCylinder_1_ExtendRetract
GlobalVariable.O_11
o_TerminalCylinder_5_ExtendRetract
GlobalVariable.O_81
Required UML structure elements
Required class
class_SortingModule
- ExtendRetract : bool
- isExtended: bool
- isRetracted: bool
- detectWorkpiece: bool
+ doExtendRetract ()
+ getExtendedRetracted ()
+ getDetectWorkpiece ()
Required objects
8 instances of class_cylinder;

Fig. 1. Elements of an IEC 61131 FBD and a UML CD for the experiment application.

A. Hypotheses and Research Questions

The hypotheses (1, 2) and research questions (Q1, Q2) derived from Section III are as follows.

1) Students trained in OO modeling show an improved modeling performance.
2) The FBD programming training improves students' programming performance.

Q1) Is programming performance in FBD related to other variables (e.g., cognitive abilities, experience, workload, and knowledge) than is OO modeling performance in $\mathrm{UML/CD + SC}$?

Q2) Is modeling performance correlated to problem-solving skills?

B. Subjects

The subjects of the study were 85 apprentices from a vocational school for production engineering in Munich with a specialization in mechatronics. The students, four entire classes (two from the second and two from the third year), were also apprentices in various companies around Munich. The average age of the subjects was 19.15 years ($\mathrm{SD} = 1.98$); 93% of the subjects were male, and only 7% were female. According to their curriculum and the results of tests of their previous programming knowledge (cf. [4]), all students were being taught programming by the procedural paradigm, with a focus on the IEC 61131-3 languages. Object-oriented programming or UML is not yet included in the curriculum for mechatronics apprentices.

C. Design of the Study

In this experiment, only the SC was used as a UML behavioral modeling technique because it is better for detailed modeling tasks than the activity diagram. A sophisticated UML code generator plug-in for PLC programming was developed to meet the tool support demand for the modeling tasks. For the IEC 61131-3, the programming language FBD, widely taught and applied, was used (see also Fig. 1). The application chosen for the experiment has similar components to be modeled/programmed (see [4], i.e., three storage elements with work pieces should be sorted into five different terminals according to their material and color) to analyze whether different variants of a component will be identified as variations of the component and already built components will be reused. A full simulation of the plant was provided for debugging and testing purposes.

The study was carried out as a field experiment with a 2×2 between-subjects design. The factor notation [39] was experimentally varied with either UML CD and SC $(\mathrm{UML/CD+SC})$ or IEC 61131-3 using FBD (61131/FBD). The second factor,

TABLE I
EXPERIMENTAL DESIGN

	Experimental groups			
Factors	Group 1 (n = 18)	Group 2 (n = 26)	Group 3 (n = 20)	Group 4 (n = 21)
1. Notation	UML/CD+ SC	UML/CD+ SC	61131/FBD	61131/FBD
2. Experience	Basic knowledge	Advanced knowledge	Basic knowledge	Advanced knowledge

expertise, was not experimentally adjusted, but varied naturally between the less experienced second-year classes and the more experienced third-year classes; see Table I. (Please note that due to this fact, advanced experience means advanced knowledge in automation hardware and IEC 61131-3/FBD, but not in UML.)

D. Procedure

Each of the four classes took part in a training set in a hybrid-learning environment (HLE), switching between computer-based and conventional instructional designs. The digital learning section consisted of a sequenced, 45-min-long recording of an ideal-type programming approach, using one of the approaches for each group and four practical training scenarios with training and instruction. An additional 22 $(\mathrm{UML/CD+SC})$ or 21 (61131/FBD) pages of documentation were provided as a training handout.

Overall, the training lasted one and a half days, with groups repeatedly performing programming and modeling tasks. During the second half-day, the subjects performed the experimental task. Before and during the training phases, and after the experimental phase, the subjects were asked to assess the training in several questionnaires.

E. Performance and Related Variables

Data were obtained for several variables assumed to be related to programming/modeling performance. These variables are the graduation and special grades in mathematics, German, automation, and mechatronics, as well as cognitive capabilities, motivation levels, challenge, and workload (single instruments are described in [3]).

Two methods were employed to obtain the performance variables.

1) Before and after the training, students took an 18-item knowledge test, requiring them to name and explain the function of components, terms, and definitions and to recognize $\mathrm{UML/CD + SC}$ or IEC 61131/FBD. The tests lasted about 4.5 min and differed in content for 61131/FBD and $\mathrm{UML/CD + SC}$.
2) A second dependent performance variable was the programming/modeling achievement itself. To obtain this value, the developed models/programs were stored and analyzed manually by two evaluators, who compared them to a full reference model (interrater reliability from $\mathrm{K} = 0.65$ up to $\mathrm{K} = 0.91$). The trainees' performance was measured as number of correctly modeled or programmed elements, and compared with respect to structure (classes or FBDs) and behavior (state charts and FBDs). These performance scales serve as performance scales for structure and behavior as well as being an overall measure.

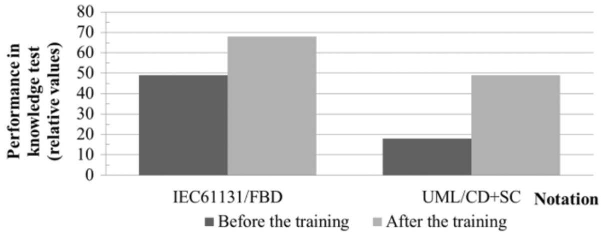

Fig. 2. Results of the knowledge tests before and after the training for beginners in IEC 61131-3.

3) To measure problem-solving skills, which should be correlated to modeling performance according to the second research question (Q2), the $\mathrm{UML/CD + SC}$ beginner group worked on a well-defined toy problem known as missionaries and cannibals (MAC). The pure processing time, after instruction and clarification of questions, was 15 min. In terms of ecological validity, Jeffries *et al.* [18] claim that the mental processes for solving the problem highly overlap with those used to solve other formal iterative problems.

F. Results

The data were analyzed using three methods. First, an analysis of variance (ANOVA) was applied to test differences between the variances of several groups in order to show whether their performance changed after the training approaches and whether their performance differed between classes due to expertise level differences. Then, correlations were computed as a measure of the relation between programming/modeling performances on different performance scales. Subsequently, differences in relations between the two software engineering approaches (see hypothesis 3) were shown by computing regression models for the programming performance for both approaches.

Hypotheses 1 and 2: In order to examine hypotheses 1 and 2, an ANOVA with the dependent variable knowledge before and after the training was carried out. The analysis examined the within factor (before and after the training) and additionally two between factors (notation and expertise). The results show that the training was highly effective since the knowledge before and after the training differed significantly $(\mathrm{F}(1,81) = 258.67, \mathrm{P} < .001, \eta^2 = 0.762)$. All participants learned from the training. Regarding the factors notation $(\mathrm{F}(1,81) = 124.43, \mathrm{P} < .001, \eta^2 = 0.606)$ and expertise $(\mathrm{F}(1,81) = 9.67, \mathrm{P} = .002, \eta^2 = 0.110)$, highly significant differences are found, which means that the UML/SC-groups and 61131/FBD-groups as well as the experienced and not experienced groups differ with respect to the knowledge tested. Both interactions (training*notation: $\mathrm{F}(1,81) = 55.79, \mathrm{P} < .001, \eta^2 = 0.408$; training*notation*expertise: $\mathrm{F}(1,81) = 8.26, \mathrm{P} = .005, \eta^2 = 0.093$) were also highly significant, indicating that the $\mathrm{UML/CD + SC}$-group had learned even more from the training because of their poor results before the training (see also Fig. 2) and their lack of prior knowledge.

Research Questions and Related Analyses: In order to give insights into the differences between the two programming/modeling approaches, the correlations were computed between the performance in programming/modeling for both

TABLE II
CORRELATIONS SEPARATED FOR 61131/FBD AND $\mathrm{UML/CD + SC}$

	61131/FBD	**61131/FBD**		**UML/CD**	**UML/SC**	
Scale / Variables	**Structure**	**Behavior**	**Overall**	**Structure**	**Behavior**	**Overall**
Cognitive abilities	r = 0.266; P = 0.092	ns	ns	ns	ns	ns
Mathematics (school grade)	r = 0.331*; P = 0.043; ß = 0.259; P = 0.067	ns	r = 0.265; P = 0.108	r = 0.557*; P = 0.001	r = 0.338; P = 0.068	r = 0.477*; P =0.008; ß = 0.666*; P < 0.001
German (school grade)	ns	ns	ns	ns	ns	ns
Automation (school grade)	ns	r = 0.366*; P = 0.020	r = 0.328*; P = 0.039	ns	ns	ns
Mechatronics (school grade)	ns	r = 0.484*; P = 0.023	r = 0.416*; P = 0.045	ns	r = 0.357*; P = 0.020	r=.328*; P = 0.034
Interest	ns	ns	ns	ns	r = 0.251; P = 0.100	ns
Cognitive demand	ns ß = -0.333*; P = 0.023	r = -0.492*; P = 0.001	r = -0.418*; P = 0.007	ns	ns	ns ß = 0.375*; P = 0.024
Frustration	ns	r = -0.480*; P = 0.001	r = -0.441*; P = 0.004	ns	r = -0.305*; P = 0.049	r = -0.285; P = 0.068
Previous knowledge in programming/ modeling	r = 0.293; P = 0.069; ß = 0.336*; P = 0.031	r = 0.492*; P = 0.001	r = 0.476*; P = 0.002	ns	ns	ns

Legend:
r: correlation coefficient (correlation between 0.30 and 0.50 are estimated as mean correlations higher values indicate strong relations;
P: level of significance (*indicating P < 0.05)
ß: regression weight (indicating the relevance of the variable for the prediction of the programming/modeling performance)

software engineering approaches and the variables listed above (i.e., grades in different subjects, cognitive abilities and motivational variables). The grades were transformed into positive values ranging from 1 to 5, with higher values indicating better results. For the overall performance measures for $\mathrm{UML/CD + SC}$ and 61131/FBD, the significant correlations $(0.000 \leq \mathrm{P} \leq 0.05)$ and tendencies $(0.05 \leq \mathrm{P} \leq 0.10)$ given in Table II were found. At this summarized level, nearly all surveyed measures of cognitive ability, motivation, and grades were at least somewhat related to certain aspects of programming/modeling performance. The only exception for summarized correlations was the grade in German. Furthermore, a certain structure can be detected when the different scales are examined. Cognitive abilities and the grade in mathematics were mainly correlated to the performance scale structure, whereas the other variables were significantly related to the scale behavior or, in the case of previous knowledge, with both scales.

Since the differences in the relations between the two approaches ($\mathrm{UML/CD + SC}$ versus 61131/FBD) were especially interesting, the correlations were also examined separately (cf. Table II). When examining 61131/FDB programming, the correlations stayed nearly the same. Differences only arise for interest, which is not meaningful.

In contrast to the 61131/FBD approach, the result for performance scales for the $\mathrm{UML/CD + SC}$ groups was not related to cognitive abilities. They were, however, more highly related to the grades in mathematics and stayed nearly the same for the grades in mechatronics. The grades in automation and German, motivational factors, and previous knowledge in modeling are not relevant for the correlations. $\mathrm{UML/CD + SC}$ performance scales show only weak correlations with subjective ratings of cognitive demand and show a mean correlation with frustration, especially for the performance scale behavior.

On the basis of these findings, it can be stated that OO modeling and FBD programming show different relations to variables such as cognitive abilities, experience, workload, and knowledge. A separate examination of the beginners in $\mathrm{UML/CD + SC}$ $(\mathrm{n} = 18)$ indicates another difference. This group shows slight correlation between the performance scale structure and the grade in German $(\mathrm{r} = 0.482; \mathrm{P} = 0.092)$. However, further examination of the various experimental groups according to Table I was not warranted because of the low number of cases in each group.

To give more insight into the differences addressed with research question Q1, regression models for the programming performance in both approaches were computed. The main regression weights (ß) were also listed in Table II. The regression model for 61131/FBD programming $(\mathrm{P} = 0.002$, explained variance: 42.6% [corrected: 33.6%]) differs from the model for $\mathrm{UML/CD + SC}$ programming $(\mathrm{P} = 0.005$; explained variance: 52.0% [corrected: 41.6%]). Again, the regression models show the differences in explaining the two approaches: Student performance in the $\mathrm{UML/CD} +$ SC groups seems to be less related to previous knowledge and cognitive abilities than does student performance in the 61131/FBD groups. Unfortunately, none of the 18 students tested found a solution to the MAC task.

V. DISCUSSION AND CONCLUSION

The first two hypotheses served as a treatment check to show that students can be trained to carry out authentic programming tasks within one and a half days. It can be stated that the training for the different programming approaches—61131/FBD and $\mathrm{UML/CD} + \mathrm{SC}$—was extremely successful when performing a task in a PLC programming environment. The created HLE, in combination with the additional handout, allowed switching between computer-based and conventional instructional designs, letting the apprentices exercise the programming/modeling tasks repeatedly. This prepared them so well for the test situation that they possessed significantly more knowledge afterwards in both programming approaches. The results presented with hypotheses 1 and 2 therefore support the reported findings of the importance of adequate time and authentic problems when teaching freshmen or beginners programming/modeling.

Q1 required examination of the prerequisites for programming/modeling. In this context, cognitive abilities, experience, motivational variables like interest and frustration, workload (cognitive demand), and knowledge were of interest. It was hypothesized and confirmed that the programming performance in 61131/FBD is related to variables other than those related to OO modeling with $\mathrm{UML/CD} + \mathrm{SC}$. The results showed that 61131/FDB programming is mainly related to cognitive abilities, mathematics, automation, mechatronics, and the subjective ratings of cognitive demand and frustration, as well as previous knowledge. In contrast, $\mathrm{UML/CD} + \mathrm{SC}$ modeling is not related to cognitive abilities and previous knowledge, but is related to mathematics and mechatronics. Regression models went in the same direction. Mathematics explains more variance of modeling performance in the $\mathrm{UML/CD} + \mathrm{SC}$ group than in the 61131/FBD group. Therefore, it is concluded that abilities determining mathematical competencies are more relevant for learning and modeling with $\mathrm{UML/CD} + \mathrm{SC}$ than for classic procedural paradigms (IEC 61131-3). This might also be due to the high taxonomy level of the cognitive skills (synthesis and evaluation) students had to demonstrate in the experimental programming/modeling task.

Another question was related to the method of preparing computer science students and the role of problem solving. Unfortunately, none of the 18 students tested on untrained and abstract problems were able to find a solution within the limit of 15 min of pure processing time. This time limit might, however, be a point of criticism, in that the MAC problem can take some time for a beginner to figure out. This should be examined in a follow-up study. On the other hand, together with the findings that mathematics played a key role in $\mathrm{UML/CD} + \mathrm{SC}$ modeling and that demanding tasks are correlated to better performance (on a high level), poor problem-solving skills might also indicate that abstraction is the key to the successful application of $\mathrm{UML/CD} + \mathrm{SC}$.

In summary, it is concluded that performance in PLC programming with $\mathrm{UML/CD} + \mathrm{SC}$ is limited and can be improved through two different factors.

1) Previous knowledge of 61131/FBD and $\mathrm{UML/CD} + \mathrm{SC}$ notations has to become more comparable. Therefore, teachers in vocational and higher technical schools will be trained for the next experiment, so that they can pass on their knowledge in the classroom to the apprentices and technicians. For experienced engineers in industry, a training session needs to be designed that includes these results and supports those engaged in an engineering task to change from pure IEC 61131-3 programming to an OO model-based software engineering in automation.
2) The participants should also be trained beforehand in problem-solving skills and abstract thinking. A procedural diagnosis of problem-solving skills (e.g., thinking aloud) with a small (sub)sample in advance could give some indication as to how the specific training might look.

Based on these results, the training exercises developed, including videos, handouts, simulations, and so on, will be revised and integrated in the 2013 offering of TUM's fifth-semester automation course and associated lab course for Mechatronics and Mechanical Engineering undergraduates. Cooperating vocational and technical schools are adapting the developed training for future courses on PLC programming, particularly since OO has just become mandatory in the curriculum for technicians in Germany.

References

[1] A. L. Ramos, J. V. Ferreira, and J. Barcelo, "Model-based systems engineering: An emerging approach for modern systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 1, pp. 101–111, Jan. 2012.

[2] *IEC International Standard: Programmable Controllers, Part 3: Programming Languages*, IEC 61131-3, International Electrotechnical Commission, 2003.

[3] K. Thramboulidis and G. Frey, "Towards a model-driven IEC 61131-based development process in industrial automation," *J. Softw. Educ. Appl.*, vol. 4, no. 4, pp. 217–226, 2011.

[4] B. Vogel-Heuser, S. Braun, M. Obermeier, F. Jobst, and K. Schweizer, "Usability evaluation on teaching and applying model-driven object-oriented approaches for PLC software," in *Proc. ACC*, Montréal, Canada, 2012, pp. 4463–4469.

[5] D. N. Ramos-Hernandez, P. J. Fleming, and J. M. Bass, "A novel object-oriented environment for distributed process control systems," *Control Eng. Prac.*, vol. 13, no. 2, pp. 213–230, 2005.

[6] K. Sacha, "Verification and implementation of dependable controllers," in *Proc. DepCoS-RELCOMEX*, Szklarska Poreba, Poland, Jun. 2008, pp. 143–151.

[7] U. Katzke and B. Vogel-Heuser, "Combining UML with IEC 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems," in *Proc. IFAC-HMS*, Seoul, Korea, Sep. 2007, pp. 90–94.

[8] D. Witsch and B. Vogel-Heuser, "PLC-statecharts: An approach to integrate UML-statecharts in open-loop control engineering—Aspects on behavioral semantics and model-checking," in *Proc. 18th IFAC World Congress*, Milan, Italy, 2011, pp. 7866–7872.

[9] E. Estévez, M. Marcos, and D. Orive, "Automatic generation of PLC automation projects from component-based models," *Int. J. Adv. Manuf. Tech.*, vol. 35, no. 6, pp. 527–540, 2007.

[10] L. Bassi, C. Secchi, M. Bonfé, and C. Fantuzzi, "A SysML-based methodology for manufacturing machinery modeling and design," *IEEE/ASME Trans. Mechatron.*, vol. 16, no. 6, pp. 1049–1062, Dec. 2011.

[11] D. Friedrich and B. Vogel-Heuser, "Benefit of system modeling in automation and control education," in *Proc. ACC*, New York, NY, 2007, pp. 2497–2502.

[12] C. M. Kellett, "A project-based learning approach to programmable logic design and computer architecture," *IEEE Trans. Educ.*, vol. 55, no. 3, pp. 378–383, Aug. 2012.

[13] C. A. Chung, "A cost-effective approach for the development of an integrated PC-PLC-robot system for industrial engineering education," *IEEE Trans. Educ.*, vol. 41, no. 4, pp. 306–310, Nov. 1998.

[14] J. Cross and P. Denning, "Computing curriculum 2001," Joint Curriculum Task Force, IEEE-CS/ACM Rep., 2001 [Online]. Available: http://www.acm.org/education/curric_vols/cc2001.pdf

[15] A. Tucker, "A model curriculum for K–12 computer science: Final report of the ACM K–12 Task Force Curriculum Committee," ACM, New York, NY, 2nd ed., 2006.

[16] B. Curtis, "Five paradigms in the psychology of programming," in *Handbook of Human-Computer Interaction*, M. Helander, Ed. Amsterdam, The Netherlands: Elsevier, 1988, ch. 4, pp. 87–105.

[17] H. N. Mok, "Student usage patterns and perceptions for differentiated lab exercises in an undergraduate programming course," *IEEE Trans. Educ.*, vol. 55, no. 2, pp. 213–217, May 2012.

[18] R. P. Jeffries, A. A. Turner, P. G. Polson, and M. E. Atwood, "A process model for missionaries-cannibals and other river-crossing problems," *Cogn. Psychol.*, vol. 9, no. 4, pp. 412–440, 1977.

[19] M. R. Lucas and D. M. Tilbury, "Quantitative and qualitative comparisons of PLC programs for small test bed with focus on human issues," in *Proc. AAC*, 2002, pp. 4165–4171.

[20] M. R. Lucas and D. M. Tilbury, "Methods of measuring the size and complexity of PLC programs in different logic control design methodologies," *Int. J. Adv. Manuf. Tech.*, vol. 26, pp. 436–447, 2005.

[21] K. Vankatesh, M. Zhou, and R. J. Caudill, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 611–619, Dec. 1994.

[22] J. S. Lee and P. L. Hsu, "A new approach to evaluate ladder diagrams and Petri nets via the if-then transformation," in *Proc. IEEE Conf. SMC*, Tucson, AZ, 2001, pp. 2711–2716.

[23] K. Thramboulidis, "A constructivism-based approach to teach object-oriented programming," *J. IEE*, vol. 4, no. 2, pp. 1–11, 2003.

[24] K. Thramboulidis, "Teaching advanced programming concepts in introductory computing courses: A constructivism based approach," in *Proc. ICEE*, Valencia, Spain, 2003 [Online]. Available: http://www.ineer.org/events/icee2003/proceedings/pdf/5045.pdf

[25] C. Oates and A. Zoitl, "Utilizing LEGO Mindstorms as a teaching platform for industrial automation," in *Proc. RiE*, 2010, pp. 31–36.

[26] S. H. Kim and J. W. Jeon, "Introduction for freshmen to embedded systems using LEGO Mindstorms," *IEEE Trans. Educ.*, vol. 52, no. 1, pp. 99–108, Feb. 2009.

[27] M. Berges and P. Hubwieser, "Minimally invasive programming courses: Learning OOP with(out) instruction," in *Proc. SIGCSE*, 2011, pp. 87–92.

[28] R. Faux, "Impact of preprogramming course curriculum on learning in the first programming course," *IEEE Trans. Educ.*, vol. 49, no. 1, pp. 11–15, Feb. 2006.

[29] M. L. Jacobson, R. A. Said, and H. Rehman, "Introducing design skills at the freshman level: Structured design experience," *IEEE Trans. Educ.*, vol. 49, no. 2, pp. 247–253, May 2006.

[30] I. Verginis, A. Gogoulou, E. Gouli, M. Boubouka, and M. Grigoriadou, "Enhancing learning in introductory computer science courses through SCALE: An empirical study," *IEEE Trans. Educ.*, vol. 54, no. 1, pp. 1–13, Feb. 2011.

[31] E. Lahtinen, J. Sajaniemi, M. Tukiainen, R. Bednarik, and S. Nevalainen, Eds., "A categorization of novice programmers: A cluster analysis study," in *Proc. 19th Annu. Workshop PPIG*, Joensuu, Finland, Jul. 2–6, 2007, pp. 32–41.

[32] A. S. Ruocco, "Experiences in threading UML throughout a computer science program," *IEEE Trans. Educ.*, vol. 46, no. 2, pp. 226–228, May 2003.

[33] I. Boticki, M. Katic, and S. Martin, "Exploring the educational benefits of introducing aspect-oriented programming into a programming course," *IEEE Trans. Educ.*, 2012, to be published.

[34] J. Schramm, S. Strickroth, N.-T. Le, and N. Pinkwart, "Teaching UML skills to novice programmers using a sample solution based intelligent tutoring system," in *Proc. 25th Int. Conf. FLAIRS*, Marco Island, FL, 2012, pp. 472–477.

[35] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, and C. Zander, "Debugging from the student perspective," *IEEE Trans. Educ.*, vol. 53, no. 3, pp. 390–396, Aug. 2010.

[36] J. Magenheim, W. Nelles, T. Rhode, N. Schaper, S. Schubert, and P. Stechert, "Competencies for informatics systems and modeling. Results of qualitative content analysis of expert interviews," in *Proc. IEEE EDUCON*, Madrid, Spain, 2010, pp. 513–521.

[37] J. Buckley and C. Exton, "Blooms' taxonomy: A framework for assessing programmers' knowledge of software systems," in *Proc. 11th IEEE IWPC*, 2003, pp. 165–174.

[38] J. Kim and F. J. Lerch, "Towards a model of cognitive process in logical design comparing object-oriented and traditional functional decomposition software methodologies," in *Proc. ACM CHI*, pp. 489–498.

[39] "Taxonomy of educational objectives," in *The Classification of Educational Goals: Handbook I, Cognitive Domain*, B. S. Bloom, Ed. New York: Longmans, Green, 1956.

[40] S. Patig, "A practical guide to testing the understandability of notations," in *Proc. 5th APCCM*, Wollongon, Australia, 2008, pp. 49–58.

[41] S. Patig, "Preparing meta-analysis of metamodel understandability," in *Proc. Workshop ESMD*, Toulouse, France, 2008, pp. 11–20.

[42] A. Gemino and Y. Wand, "A framework for empirical evaluation of conceptual modeling techniques," *Requirements Eng.*, vol. 09, pp. 248–260, 2004.

Birgit Vogel-Heuser (M'04–SM'12) graduated in electrical engineering and received the Ph.D. degree in mechanical engineering from the RWTH Aachen, Aachen, Germany, in 1991.

She worked for nearly 10 years in industrial automation in the machine and plant manufacturing industry. After holding different chairs of automation in Hagen, Wuppertal, and Kassel in Germany, she has been Head of the Automation and Information Systems Institute, Technische Universität München, Munich, Germany, since 2009. Her research work is focused on modeling and education in automation engineering for hybrid process and heterogeneous distributed and intelligent systems using a human-centered approach.

Prof. Vogel-Heuser is member of the GMA (NMO IFAC). She has received the Special Award of the Initiative D21 Women in Research in 2005, the Borchers Medal of the RWTH Aachen in 1991, the GfR Sponsorship Award in 1990, and the Adam Opel Award in 1989.

Martin Obermeier received the Dipl.-Ing. degree in mechanical engineering (focus on control theory and information technology) from Technische Universität München (TUM), Munich, Germany, in 2009, and is currently pursuing the Ph.D. degree at the Institute of Automation and Information Systems, TUM.

Mr. Obermeier does research on the design and evaluation of model-based programming software for automation systems in regard to model quality, modularity aspects, and usability.

Steven Braun received the Dipl.Ing. degree in mechanical engineering (focus on mechatronics and information technology) from Technische Universität München (TUM), Munich, Germany, in 2008, and is currently pursuing the Ph.D. degree at the Institute of Automation and Information Systems, TUM.

Mr. Braun does research in the design and evaluation of model based programming software for automation systems.

Kerstin Sommer received the Diploma degree in psychology (focus on human factors and statistics) from Universität Regensburg, Regensburg, Germany, in 2007, and is currently pursuing the Ph.D. degree at the Institute of Experimental Psychology, Universität Regensburg.

From 2007 to 2010, she was with the Institute of Experimental Psychology, Universität Regensburg. Currently, she is with the Institute for Automation and Information Systems, Technische Universität München, Munich, Germany. Her research interests are in the area of human factors methods and usability design.

Fabian Jobst received the diploma degree in psychology (focus on instructional design for gifted students) from Universität Würzburg, Würzburg, Germany, in 2008, and is currently pursuing the Ph.D. degree in psychology at Pädagogische Hochschule Weingarten, Weingarten, Germany.

His research interests are instructional designs and digitally augmented videos.

Karin Schweizer graduated in psychology and informatics and received the Ph.D. degree in cognitive psychology from Universität Mannheim, Mannheim, Germany, in 1996.

Since 2002, she has held visiting professorships at various German universities. Since 2010, she has been Head of the Department of Psychology, Pädagogische Hochschule Weingarten, Weingarten, Germany. Her research interests deal with different methods of technology-based learning.