Cogn Tech Work (2015) 17:417–430
DOI 10.1007/s10111-014-0307-x

ORIGINAL ARTICLE

# Towards a taxonomy of errors in PLC programming

**Kerstin C. Duschl · Denise Gramß · Martin Obermeier · Birgit Vogel-Heuser**

Received: 14 January 2014 / Accepted: 18 September 2014 / Published online: 27 September 2014
© The Author(s) 2014. This article is published with open access at Springerlink.com

**Abstract** Based on previous studies on programming errors and their causes, the presented paper investigates errors that application engineers in the area of machine and plant automation make while creating either function block diagrams, plcML [an adaption of the unified modelling language (UML)] or modAT4rMS code (a newly developed modelling language that adapts and combines aspects of UML and SysML). A laboratory-based study with 52 mechatronics apprentices and electrical engineering technicians with knowhow in manufacturing system design but comparably undeveloped programming skills has been conducted, in which the subjects' errors and think-aloud statements during code creation were recorded. In a subsequent step, these data have been analysed by the cognitive causes of the coding errors applying the skill-rules-knowledge framework. As a result, a taxonomy of errors is presented. Results indicate that most of the errors in the subjects' code are due to insufficient understanding of the notation's syntax, problems with the rules of encapsulation, the creation of modules and finally with the creation of variants and aggregations, which are all located at the rule-based level. Errors at the skill-based level mainly occurred during behavioural modelling with modAT4rMS. It is argued that the provided insights can be used for improving education on programmable logic controller (PLC) languages and for the design of tools that support PLC programmers at detecting and fixing errors within their code.

K. C. Duschl · D. Gramß · M. Obermeier · B. Vogel-Heuser (✉)
Chair of Automation and Information Systems, Technische Universität München, Boltzmannstr. 15, 85747 Garching, Germany
e-mail: vogel-heuser@ais.mw.tum.de

**Keywords** Automation · Errors · Human factors · Object-oriented methods · Software engineering

## 1 Introduction

The correctness of programming code is a critical component of its quality. However, conventional man–machine interaction methods rather focus on aspects like learnability and efficiency of a programming language instead of error proneness (Ko and Myers 2005).

After all, it definitely makes sense to deal with the latter, as even relatively simple-structured programming languages like HTML still are so complex that they provide a variety of options for syntax errors, runtime errors and bugs, i.e. unintended or exceptional behaviours (Blackwell 2002; Park et al. 2013). In experiments on so-called programmable logic controller (PLC) programming languages, which are used for automation of manufacturing systems and are subject of the study described below, errors usually are in the double-digit per cent range (Braun 2013; Hajarnavis and Young 2008). For that reason, a subsequent code testing and debugging is essential. According to Ko and Myers (2005), software engineers spend between about 70 and 80 % of their time detecting, diagnosing and repairing software problems, with an average software bug taking 17.4 h to fix. Thus, software errors are a significant cost factor. In automation technology, this is even more pronounced as the debugging cannot be done at the desk, but must take place at the construction site.

On the other hand, learning also happens through trial and error—assuming the novice has the opportunity to recognize the errors he made. In order to be able to provide better support during this process, both to novices (apprentices, students) and experienced programmers, it is worthwhile to examine, which errors are common (or even particularly frequent) in which modelling language and which cognitive causes they probably result of. So far, mistakes that are made while using programming languages for PLCs have been hardly investigated. This can be stated to constitute an important gap in the literature. One of the few exceptions is described by Hajarnavis and Young (2008), who conducted an experimental study on "right first time" rates and completion time during process modification using four different PLC notations. However, they only named some easily observable subjective "main problems", but did not perform any objective analysis on that topic.

The study presented in the following pursues three main objectives: first, to identify the errors programmers make while writing code in FBD and plcML (Witsch and Vogel-Heuser 2011) compared to the newly developed notation modAT4rMS (Braun 2013). Second, to examine the cognitive origins of these errors (skill-based, rule-based or knowledge-based errors), and third, to utilize the knowledge gained to create pedagogically superior tools.

All notations in this study provide a possibility of code/model encapsulation, which means a construct that allows the bundling of data with the methods or functions operating on that data. Such encapsulation constructs are a prerequisite for reusability. Function block diagram (FBD) is a common graphical programming language for PLCs that is standardized in the worldwide industrial programming standard IEC 61131-3 (International Electrotechnical Commission 2003). The IEC 61131-3 defines five languages, two textual languages and three graphical languages including FBD. This standard is accepted and widely used by software engineers to specify the software part of industrial automation systems (Thramboulidis and Frey 2011). These five languages of the IEC 61131-3 are suitable for different application types; FBD is mostly used for interlocking, reusable functions and communication. Its primary concept is data flow from inputs to outputs through function blocks or functions that offer a wide range of signal operations. A clean encapsulation with FBD is possible, but is easily disturbed by the use of global variables, which leads to poor reusability of modules, as described in Zoitl and Vyatkin (2009). In order to provide an elemental basis of reuse, IEC 61131-3 software can be structured into function blocks (FBs). FBs encapsulate the structure and behaviour programming of a collection of elements used in automation projects (Thramboulidis and Frey 2011).

In contrast to FBD, plcML uses the object-oriented (OO) extension of the IEC 61131-3 specification adapting classes and methods, interfaces and inheritance from the unified modelling language (UML) for object-oriented PLC programming. The object-oriented paradigm encourages a modular design by different relation types, such as association among classes and inheritance for hierarchical structuring. With the plcML class diagram, the structure of automation control software can be modelled and the understanding of the interdependencies of components can be supported. The plcML state chart diagram offers states and transitions between them to model the behaviour of automation systems. The assumed benefits of OO programming are among others "a more efficient code reuse and increased safety and stability of software" (Vyatkin 2013).

Finally, modAT4rMS is a newly developed modelling language described by Braun (2013) that adapts and combines aspects of UML and SysML with the goal to simplify OO PLC programming both for novices and experts by providing a less abstract structure notation in comparison to class diagrams. The modAT4rMS notation uses an object-centred structure diagram. In order to clearly visualize the connection between structure and behaviour diagram, an adapted state chart diagram shows only the accessible objects and their functionality according to the prior defined structure model.

This paper is structured as follows: Sect. 2 reviews prior research on programming errors in general and the difficulties of PLC programming in particular. Section 3 describes a preliminary study on modelling errors in UML and their results. Section 4 outlines the main study and their methods. Section 5 describes detailed examples of the observed coding errors and presents a taxonomy of the errors that were found. Finally, Sect. 6 discusses the findings in relation to education and for designing tools to support both apprentices and technicians.

## 2 Related work

A considerable amount of literature is dealing with the analysis and description of various types of coding errors made by programmers, with their strategies to fix them and the role of errors during the learning phase of code programming. However, these studies usually deal with classical programming languages (such as Pascal or LISP) and not with PLC programming languages. Moreover, they are often quite old: the majority of the experiments took place in the 1980s, before graphical user interfaces and object orientation were common and sometimes the experiments even analysed programming languages that became antiquated by now.

An example for these studies is, among others, Anderson and Jeffries (1985), who analysed errors of novices (students) using LISP functions. They showed that error frequency is even increased "by increasing the complexity of irrelevant aspects of the problem" and that errors mainly result from "a loss of information in the working memory representation of the problem and when the resulting answer still looks reasonable" (Anderson and Jeffries 1985). That means the errors made are rather the result of slips (e.g. forgotten brackets) than misconceptions!

Another experiment was conducted by Panko (1998) and Panko and Sprague (1999), who tested students' spreadsheet performance. They found that 54 % of the occurred errors were due to omission and 43 % were logic errors that result from a mistake in reasoning. Mechanical errors (i.e. typographical errors), however, were almost nonexistent.

According to an analysis of debugging anecdotes of industry experts, conducted by Eisenstadt (1993), the biggest error causes in COBOL, Pacal, C and Fortran programs were "memory overwrites" and "vendor-supplied hardware or software faults" (e.g. a buggy compiler) (Eisenstadt 1993), together accounting for more than 40 % of the problems occurred.

On the other hand, an error classification by Youngs (1974), which is based on a study with 42 subjects using a variety of different programming languages came to the conclusion that novices have the biggest difficulties regarding the semantics of a language, while experts show an nearly equal number of syntax errors, semantic errors and logic errors.

In a study by Perkins and Martin (1986), who interviewed high school students, "fragile" (i.e. partial, inert, misplaced, conglomerated) knowledge of the language syntax (BASIC in this case) was the cause for most of the students' errors.

An extensive review of further user-based experiments on software errors is given in Ko and Myers (2005).

As far as PLC programming is concerned, existing studies mainly focus on a comparison of the performance in code creation, but not on their cognitive causes. Vogel-Heuser et al. (2012, 2013) conducted an extensive laboratory study with 85 apprentices from a vocational school for production engineering in Munich with a specialization in mechatronics comparing UML and FBD. No significant performance differences were found between the two languages though. This finding was confirmed by Braun (2013), who tested 168 apprentices that handled the same task using either UML, FBD or modAT4rMS. However, those subjects, who used modAT4rMS, performed significantly better than those using FBD or plcML.

However, there are some studies on UML that focus on knowledge of the problem domain instead of error measurement. Siau and Tian (2001), for example, applied the goals, operators, methods and selection rules (GOMS) technique to evaluate the diagramming techniques in UML. Siau and Loo (2006) used cognitive mapping to study difficulties in learning UML. And Purao et al. (2002) utilized the think-aloud method (Ericsson and Simon 1984) to understand the intentions of two developers while using UML.

## 2.1 Filling the gap: deficits of current evaluation studies

Despite the amount of literature on programming errors, we still lack a detailed understanding of the errors people are likely to make when creating code in FBD, plcML or modAT4rMS. All we know is that the latter appears to be less prone to errors (novices' correct mean modelling performance $M = 0.60$ for structure and $M = 0.42$ for behaviour), while the novices' correct performance rates in FBD ($M = 0.25$ for structure and $M = 0.15$ for behaviour) and plcML ($M = 0.21$ for structure and $M = 0.16$ for behaviour) seem to be comparable (Braun 2013). The latter is confirmed by Vogel-Heuser et al. (2012, 2013), who compared performance rates in FBD ($M = 0.86$ for structure and $M = 0.43$ for behaviour) and UML ($M = 0.86$ for structure and $M = 0.45$ for behaviour). The performance differences between Braun (2013) on the one hand and Vogel-Heuser et al. (2012, 2013) on the other hand are probably due to differences in task difficulty: the task used by Braun (2013) is far more complex.

In order to provide an empirically based reasoning, why this newly developed notation leads to a superior performance compared to the other two languages and what is particularly difficult in each language, qualitative and quantitative data are needed on (a) what errors people make when creating code in FBD, plcML and modAT4rMS, as well on (b) what are the causes of these errors.

Another deficit of existing studies that should be addressed is that the defined classifications usually do not actually describe pure software errors, but often mixes them with runtime faults, runtime failures and cognitive failures. For this reason, Ko and Myers propose to differentiate between four salient aspects of software errors:

1. The surface qualities of the error (syntactic or notational anomalies in a particular code fragment, as, for example, typos or oversights).
2. The cognitive causes of software errors (e.g. lack of knowledge about language syntax, data types, attention issues such as forgetting or a lack of vigilance, and, strategic problems like unforeseen code interactions or poorly designed algorithms).
3. The programming activity in which the cause of the software error occurred (e.g. during specification activities or during algorithm design activities).
4. The type of action that led to the error (creating, reusing, modifying, designing, exploring or understanding).

One possible problem with classifications that are based only on the surface qualities is that an incorrectly coded algorithm can have many causes: it might be due to "an invalid understanding of the specifications, a lack of experience with a language construct, misleading information from a debugging session or simply momentary inattention" (Ko and Myers 2005). Depending on the cause, the resulting error must be approached in a completely different way. The current contribution provides an analysis of errors in modelling task performance. Beyond an error classification based on surface qualities, the use of the think-aloud technique enables a deeper understanding of underlying cognitive processes during task performance.

### 2.2 Empirical evaluation concept

In order to overcome the deficits described in the previous section, we suggest a classification scheme focusing on the underlying cognitive mechanisms of human error. Thereby, we suggest adapting a hierarchical model first proposed by Rasmussen (1983) and further elaborated by Reason (1990). According to that framework, human behaviour is organized in terms of cognitive effort and is based either on skills, rules or on knowledge. Rasmussen (1983) proposed skilled-based behaviour as sensory motor performance in activities, mostly without conscious control. The behaviour is based on highly integrated patterns which are predominantly automated in human behaviour. Subroutines in familiar situations are controlled by rules or procedures which previously derived, e.g. from experience, persons' knowhow or instructions. The rule-based performance is goal-oriented, but the goal is not even explicitly formulated. It is rather implicit in the situation. The distinction of skilled-based and rule-based performances is not quite clear. It depends on the level of training and attention of the person. In unfamiliar situations in which knowhow or rules are not available, the performance is controlled on a higher conceptual level. The behaviour is knowledge based. The goal is explicitly and based on the analysis of the environment and the overall aims of the person. This level contains functional reasoning, the development of an internal structure in terms of a mental model and a plan to achieve the explicitly formulated goal.

Reason (1990) assumes that routine actions in a familiar environment are carried out in a highly automated manner on the skill-based level. If a problem occurs, the human being changes to the rule-based level, where he searches (under consideration of local state information) for familiar patterns, in order to apply a stored IF (situation) THEN (action) rule. Ko and Myers (2005) compare these rules with programming plans (Spohrer and Soloway 1986) underlying the development of programming expertise (Davies 1994). Only, if the rule cannot solve the problem, a higher level analogy is searched on the knowledge-based level. If none could be found, finally more abstract relations between structure and function are analysed and the human being subsequently tries to infer diagnoses and to formulate corrective actions.

According to Reason (1990), two types of error can occur within this system: on the one hand, monitoring errors, which are not intended by the human being but happen incidentally, and on the other hand, mistakes, which are the result of a properly executed but deficient plan.

Monitoring errors are failures on the skill-based level. They happen, because the progress of action (which are especially important in the proximity of critical decision points) is omitted or does not happen timely (e.g. because of internal distraction). These unintended failures are either due to inattention (e.g. typing errors) or due to memory problems (e.g. omissions, where planned intermediate steps of long action sequences are skipped) (Reason 1990).

Mistakes, however, happen on the rule- or knowledge-based level. In the case of a rule-based failure, they result either from the misapplication of a good rule (which is useful elsewhere) or from the application of a (substantial) bad rule. An example of the first is the choice of an appropriate "while" loop for a problem (Shackelford and Badre 1993); examples of the latter are simple syntax errors and malformed Boolean logic. According to Ko and Myers (2005), these bad rules derive from "learning difficulties, inexperience or a lack of understanding about a particular program's semantics".

Knowledge-based failures typically arise from a lack of knowledge about the situation (especially, if the task is very complex), a lack of understanding of causal relationships or an incomplete/inappropriate mental model of the problem space (Reason 1990).

The rule-based and knowledge-based performances are not quite distinct. The perception of information is generally not dependent on the form of representation. Rather the context of information and expectations of the perceiver are necessary (Rasmussen 1983).

Within the context of programming and modelling, the differentiation of intended versus unintended action seems to be very interesting, as experienced programmers may solve modelling tasks in such an automated manner that inattention or memory failures lead to unintended faulty results. Sometimes, the reason of a particular error can be identified quite easily when analysing the final model or programming code. Unfortunately, this is not always possible.

As a solution, we propose to *observe* the subjects during the code creation process in combination with the *think-aloud method* (Ericsson and Simon 1984; Boren and Ramey 2000). Furthermore, we propose to record the progress of their code together with self-reports of the subjects' thoughts, the goals they pursue, their decision-making and the rationale behind their actions. Using this method, it is possible to obtain both the directly observable erroneous behaviour and the corresponding type of cognitive breakdown as described by Reason (1990).

The goal of this analysis is to examine the different kinds of programming errors according to Rasmussens SRK model. The prior studies (e.g. Vogel-Heuser et al. 2012, 2013) lack analyses of reasons for performance differences in structure and behaviour modelling tasks and different notations (FBD vs. UML). Additionally, the comparison of the errors occurring while using FBD, plcML and modAT4rMS should reveal reasons for difficulties in modelling with the specific notations. Knowledge about the origin of errors in the modelling tasks is expected to deduce implications for education as well as tool design with the purpose to decrease errors in modelling tasks. A deeper analysis of performance deficits in modelling with specific notations should fill the gap of evaluation studies in this specific domain.

## 3 Preliminary study: UML modelling failures and their causes

The starting point for the survey presented in the following was a study that examined UML modelling failures made by mechanical engineering students and their reasons. The objective then was to estimate the benefit of model-based engineering with UML in machine and plant automation and to decide on appropriate support methods. Moreover, the impact of the modelling task order (structural modelling first versus behavioural modelling first) has been examined, as Robins et al. (2003) and Mayrhauser and Vans (1997) suspect it to have an effect on the subjects' performance.

### 3.1 Experimental design

In total, 102 subjects (89.1 % male) with an age between 18 and 27 years participated in this study. All subjects were mechanical engineering students in their second year of education at the Technische Universität München (TUM), who attended a small group course on "fundamentals in informatics".

The study took place in a total of ten parallel courses at the TUM and was part of a practice course the subjects attended. The subjects were introduced to the topic during two 90-min lecture sessions. During the experiment, the subjects had to create UML diagrams of the structure and behaviour of a given sorting system that should have been able to handle two different work piece types and sort each type in one of two different storages using given sensors in combination with two cylinders. The subjects had 25 min to complete this task, whereby 70 of the students started with structural modelling and the remaining 32 students started with behavioural modelling.

The performance assessment was done using a proven coding system, in which points were awarded for correct representations of relevant features, but no points were deducted for errors or incomplete/missing data. A total of 46 points were achievable, including 22 for structural modelling and 24 for behavioural modelling.

A week later, when the UML models had been evaluated and the perfect solution was presented and explained, the subjects were either interviewed or had to fill out a questionnaire on the assumed reasons of their failures (both in general and specific). During the specific questions, they first were asked, if a particular error reason (e.g. poor concentration) applied, and, if yes, which of the occurred failures were the result of that problem. The statements made in the questionnaires and interviews were transferred to a data matrix and supplemented with the objective behaviour and structural modelling performance measures of the respective subject. In a second step, the subject's linkages between particular failure causes (e.g. lack of time) and the specific failures attributable to them were checked.

### 3.2 Results

Considering the UML models, the average participant reached 19.97 out of 46 points (SD = 9.1819), while the performance gap between the participants ranged from 2 to 42 points.

According to 72 % of the subjects, lack of time for processing the task was part of the problem. Another 56 % complained about difficulties to "translate" their mental model of the sorting system into a correct UML model, and 31 % declared to have made some of the errors due to distraction from the task. Moreover, 25 % indicated that they have misunderstood parts of the task and thus have formed an incorrect mental model of the sorting system and 21 % reported to have overlooked essential aspects of the task.

Unfortunately, 24 % and 19 % of the lacking points (for structural modelling and behaviour modelling), respectively, could not be linked by the subjects to any of the tested variables.

A regression analysis including demographics (field of study, age and gender), the subjective failure causes and the modelling order as independent variables explained about 40 % of the observed scatter in the performance data. Highly significant were the factors "modelling order" (behaviour first was better than structure first), "lack of time" and "translation problems" (from the mental model to a UML model) on structural and behavioural modelling performance.

### 3.3 Consequences/constraints for further studies

The results indicate that (besides the UML modelling order and lack of time) difficulties to "translate" the subject's mental model of the sorting system into a UML model were the main reason for the shortcomings in the subjects' performance. Thus, it can be concluded that in future education of UML, the focus should be on these "translation" problems.

In order to reduce the percentage of failures that could not be linked to any of the tested variables, an experimental procedure applying observation and think aloud seems to be promising. Further insights could be gained by analysing the modelling process with the help of tools like key logging, eye tracking and/or video recordings.

## 4 Experimental design

In order to understand the errors that PLC programmers make when writing code using either plcML, modAT4rMS or FBD and being able to understand their causes, it was necessary to collect detailed observations in a laboratory-based study. A total of 52 subjects had been observed and recorded using a think-aloud protocol as they completed a given PLC programming task using the language they had been trained 2 days before (plcML, modAT4rMS or FBD). Afterwards, the audio and screen capture data were analysed via a process similar to open and axial coding from grounded theory (Strauss and Corbin 1998) in order to construct a taxonomy of errors.

### 4.1 Participants

The study involved 52 subjects with the youngest being 17 and the oldest 27 years ($M = 20.5$, $SD = 2.797$). Forty-seven subjects were male—only five were female. All subjects received 2 days of training in either plcML (eight subjects), modAT4rMS (24 subjects) or FBD (20 subjects) 2 days prior to the test described here.

Twenty-one subjects were mechatronics apprentices in their second year at a vocational school in Munich, 13 subjects were in their third year at the same school, and 18 were electrical engineering technicians, who took part in a further training at a school for state-certified technicians in Munich.

Participation in the study was carried out on a voluntary basis and was not remunerated.

### 4.2 Instruments

The subjects were tested individually in a quiet room at their school in single sessions lasting approximately 45 min. The coding task was carried out using a computer with the necessary programming/modelling software (prototype for modAT4rMS and Codesys for FBD and plcML), a 24 inch screen, and the progress was audio-recorded and video-recorded with the screencasting software Camtasia Studio.

*Programming/modelling performance* was assessed with a coding task that was identical for all three languages and was pretested by the experimenters to ensure that it could be completed in about 15–20 min. The task was handed out to the subjects at the beginning of the session as a printed instruction containing multiple subgoals as well as an image and a table depicting the configuration of the system (see Fig. 1; Table 1).

The subjects had to programme the sorting section of a bottle-storing system, where only one conveyor belt (each with a motor driving the conveyor in one direction) should

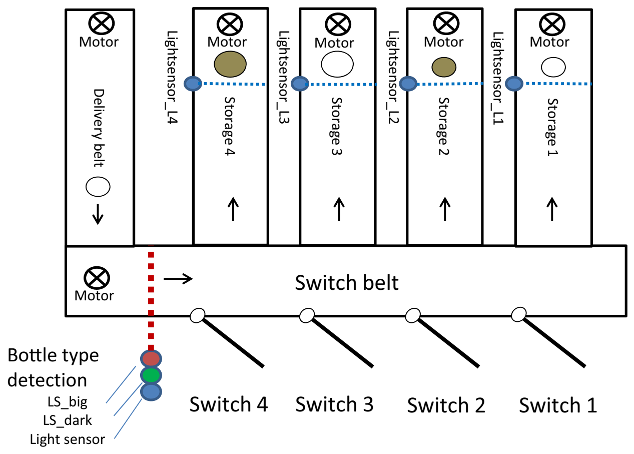

**Fig. 1** Configuration of the system that is to be programmed/modelled by the subjects

**Table 1** Logic of the bottle sorting

| | Sensor big—small | Sensor bright—dark | Sensor bottle provided | Sensor storage |
|---|---|---|---|---|
| Bottle small, white | 0 | 0 | 1 | 1 |
| Bottle small, brown | 0 | 1 | 1 | 2 |
| Bottle big, white | 1 | 0 | 1 | 3 |
| Bottle big, brown | 1 | 1 | 1 | 4 |

be active at a time. One bottle was already set on the delivery belt. That belt had to run for at least 5 s in order to make sure that the bottle is on the course tape. On the switch belt, the bottle first moved to the bottle-type detection that consisted of three sensors: one that generally recognized that a bottle passes, one that distinguished bright from dark material, and finally, one that detected whether the bottle is big or small. Thus, the bottle type could be detected. After recognition of the bottle type, the corresponding switch should be activated by a binary control signal (transport to the corresponding storage belt vs. let pass). Afterwards, the belt should run for a while according to the provided switch in order to transport the bottle to the switch.

The transport time from detection to switch 1 should be 8 s, 6 s to switch 2, 4 s to switch 3 and 2 s to switch 4. After the bottle had been transported to the switch, the switch belt should stop and the corresponding storage belt should eventually carry away the bottle. The bottle should be considered sorted when the light sensor at the end of the storage belt was activated. Finally, a new bottle should be automatically set to the delivery belt, the switches should be reset, and the process should begin again.

For the *identification of cognitive error causes,* the subjects were asked to think aloud during the whole task completion. Thereby, their statements were recorded and saved as an audio file parallel to the screencasting. Towards the end of the test, the subjects also were taught about errors they made during task completion and were questioned about their assumptions regarding the causes of their errors.

## 4.3 Procedure

After the subject had arrived in the test room and made brief statements on his/her demographic characteristics, he/she received the printed instruction of the coding task and was asked to complete it to the best of his/her ability using the respective language that he/she had learned in the previous training. That is, eight subjects created a plcML model, 24 subjects worked with modAT4rMS, and 20 subjects were using FBD. Moreover, the think-aloud protocol was explained and the subjects were continuously encouraged to express their thought processes as they completed the task ("Go on, tell me! What are you doing at the moment?"). The disturbance by the experimenter was kept to a minimum.

A maximum of 45 min was provided for the task. After the task had been completed (or the time limit exceeded), the experimenters asked follow-up questions about the errors that have been made in order to clarify the subject's understanding and intent.

The sessions were audio-recorded and video-recorded with the subjects' consent.

## 4.4 Data analysis

An analysis schema was not developed previously. For data analysis, the methods of open and axial coding from the ground theory were applied. Rather open coding concerned with identifying, naming, categorizing and describing of phenomena found in data. The first step of data analysis results in abstract categories and concrete ones to describe errors. In the next step, after open coding, axial coding is a procedure to put data together in new ways by connections between several categories (Strauss and Corbin 1998). Heuristics for error classification in different levels (skill, rule and knowledge) are described.

A total of more than 40 h of video and audio data were analysed by two researchers in three iterative rounds. No predetermined codebook had been applied. Instead, the inventory of errors was developed during the coding by using an inductive, data-driven process. A common conceptual vocabulary was ensured by the cooperation and permanent discussion between the two coders.

In the initial round of coding, every occurrence of an error was noted and described together with the time stamp of the occurrence and with the number of missed points relative to the total number of points that were achievable. Thereby, errors were defined as code with invalid syntax or as code that resulted in output that was not desirable according to the task (see Youngs 1974). Errors that were resolved by the subjects themselves during the task completion were noted, but were not classified as errors. However, they sometimes led to a substantial time delay.

In the next round, the identified errors were classified according to the evolving coding scheme, which was inspired by the skill-rule-knowledge framework first proposed by Rasmussen (1983) and further elaborated by Reason (1990) (see Chapter 1 for details). Each error was considered to be the result of a cognitive breakdown on either the skill-based, rule-based or knowledge-based level and consequently was assigned to one of them. In doing so, the researchers relied both, on observed behaviours, the subjects' verbalizations while creating code, as well as their statements in the follow-up discussion on the errors they had made. For instance, a misapplication of the timer function, suggesting rule-based behaviour, or a typographical error occurring only once, would be typical for a skill-based breakdown. The heuristics used for the classification of errors occurring at the skill-, rule- and knowledge-based levels of performance are based on suggestions by Ko and Myers (2005) and are outlined in Table 2.

**Table 2** Heuristics for the classification of errors according to Ko and Myers (2005)

| | Skill | Rule | Knowledge |
|---|---|---|---|
| Type of activity | Active execution of routine, practiced actions in a familiar context | Detection of a deviation from the planned for conditions | Execution of unpracticed or novel actions |
| | Internal focus on problem solving, rather than executing the routine actions | Seeking of signs in the environment to determine what to do next | Comprehending, hypothesizing or otherwise reasoning about a problem using knowledge of the problem space |
| Situation, when breakdown happens | Interruption by an external event | Taking of the wrong action | Decision-making without consideration of all courses of action or all hypotheses (biased reviewing) |
| | Delay between an intention and a corresponding routine action | Missing of an important sign | False hypothesis (confirmation bias) |
| | Performing of routine actions in exceptional circumstances | Information overload | Seeing of a nonexistent relationship between events (simplified causality) |
| | Performing of multiple, similar plans of routine action | Acting in an exceptional circumstance | Illusory correlation or failure to notice a real correlation between events |
| | Missing of an important change in the environment while performing routine actions | Missing of ambiguous or hidden signs | Inattention to logically important information when making a decision (selectivity) |
| | Attention to routine actions and false assumption about their progress (omission, repetition) | Acting on incomplete knowledge | Not considering logically important information that is difficult to recall |
| | | Acting on inaccurate knowledge | Overconfidence about the correctness and completeness of one's own knowledge |
| | | Use of an exceptional, albeit successful rule from past experience | |

In the third round of analysis, the derived categories in the classifications were further refined and/or combined to umbrella terms. Moreover, the classification was specified resulting in a detailed taxonomy of error types.

## 5 Results

Results are considered separately for behavioural and structural modelling. The first analysis contains the percentage of the correct code solved suitable for task requirements. Accordingly, the remaining percentages are errors, technical bugs or were classified as unclear. Secondly, these errors are analysed in terms of skill-based, rule-based and knowledge-based error levels.

First, an analysis of the performance achieved in processing the structural and behavioural part of the task was carried out, comparing the three languages. In the structural modelling, participants modelling with plcML completed 65.75 % on average (SD = 36.26, range from 0 to 100), those who used modAT4rMS completed 92.37 % (SD = 15.40, range from 22 to 100), and those programming with FBD achieved 76.55 % (SD = 24.45, range from 21 to 100). Performance at the behavioural modelling was 10.87 % on average (SD = 8.77, range from 0 to 25) when using plcML, compared to 74.75 % (SD = 21.74, range from 0 to 100) and 60.25 % (SD = 32.13, range from 2 to 96) with modAT4rMS and FBD, respectively (see Table 3).

Both for the structural and the behavioural modelling, single factor variance analyses reveal a highly significant performance difference depending on the language used. ModAT4rMS leads to better structural models than FBD and plcML ($p < 0.01$), while the behavioural part is modelled worst with plcML, while FBD and ModAT4rMS were comparable ($p < 0.001$).

In the following, some example cases are described in detail, representing errors at the skill-based, rule-based and knowledge-based level. Afterwards, a taxonomy of errors is presented that is based on all of the observed errors.

### 5.1 Examples of errors at the three levels

#### *5.1.1 Skill-based errors*

Participant uml3cip11, a 21-year-old male trainee in his second year of education, working on the behavioural part of the plcML modelling once inadvertently types

```
binAktor:status.an: = 1 und 0
```
instead of
```
binAktor:status_an: = 1 und 0
```

**Table 3** Mean per cent (with standard deviation) of correct code in the structural and behavioural modelling of the task when using either plcML, modAT4rMS or FBD

| | Structural modelling % correct | Behavioural modelling % correct |
|---|---|---|
| plcML | *M* = 65.75 | *M* = 10.87 |
| | SD = 36.26 | SD = 8.77 |
| modAT4rMS | *M* = 92.37 | *M* = 74.75 |
| | SD = 15.40 | SD = 21.74 |
| FBD | *M* = 76.55 | *M* = 60.25 |
| | SD = 24.45 | SD = 32.13 |

indicating a slip of the finger as the two keys lie directly next to each other.

Participant mod6pru03, a 24-year-old male technician using modAT4rMS, forgets the reset of the storage variables during the behavioural modelling and types

`Lager ==0`
instead of
`Lager: = 0`

while the remaining assignments are mostly correct before and after this instance.

And, finally, participant fup2pru08, a 18-year-old male trainee in his second year of education, accidentally links the sensor queries for different work pieces without the required SR function blocks with TON function blocks with the respective waiting periods. After briefly examining his code in the follow-up interview, he realizes the error himself and tells the experimenters what the correct solution would have been.

### *5.1.2 Rule-based errors*

Participant uml3cip15, a 21-year-old male in his second year of education using plcML, creates an aggregation of binary sensors for work piece recognition in the binary actuator, which points to a lack of understanding in encapsulation and the forming of modules as the aggregation does not correspond to the existing hardware structure.

In modAT4rMS, subject mod4cip11, who is a 19-year-old male in his third year of education, creates a bottle detection with the type binSensoren. Later, he declares that he does not know the difference to binSensor, which indicates a lack of understanding in encapsulation, too. In particular, the relationship between signal and method seems to be unclear.

An example of a rule-based error in FBD is provided by subject fup5pru18, a 17-year-old male in his third year of education: he creates function calls in the motor under the `Var_output` category instead of the `Var_input` category, which can be interpreted as incomplete understanding of the notation's syntax.

### *5.1.3 Knowledge-based errors*

Participant uml3pru10, a 22-year-old male in his second year of education, modelled the task with plcML without a delay time after the activation of the switch belt. On questioning, he stated that he had difficulties to understand the process, i.e. he was unable to really understand the task.

The same happened to subject mod1pru18, a 20-year-old male, who also was in his second year of education, but used modAT4rMS. He, too, had difficulties with the understanding of the process, resulting in a faulty order of the function Lagerband start, where the conveying starts only after checking, if the sensor is on 1.

In FBD, no knowledge-based errors have been noticed.

## 5.2 Classification of the errors

As described in Chapter 3, the errors were classified via open and axial coding that was inspired by the skill-rule-knowledge framework considering both the errors' context and the subjects' statements. The analysis revealed that by far the most errors occurred at the rule-based level, while errors on the skill-based and on the knowledge-based levels were quite rare.

Points that were missed due to lack of time were classified in the category "unclear", since it cannot be determined with certainty whether these parts would have been properly resolved, whether the subjects had more time to process the task, or which errors would have occurred. In general, lack of time is caused by prior inefficiency in modelling or by very long thinking pauses throughout the processing of the task.

Errors due to technical failure of the programming environment were recorded only in FBD, when a bug occurred during programming with CoDeSys.

Table 4 provides an overview on the percentage of skill-based, rule-based and knowledge-based errors in the structural and behavioural part of the task when using either plcML, modAT4rMS or FBD.

Although being quite frequent, mixed-design variance analyses (with notation as a between-subject variable and task—structural vs. behavioural modelling—as a within-subject variable) do not reveal significant differences in the occurrence of rule-based errors depending on the notation used. The same is the case with the comparably exotic knowledge-based errors and with errors due to a technical failure (bug) in the modelling software. However, rule-based and knowledge-based errors show a significant task effect being more frequent during behavioural modelling than during structural modelling.

**Table 4** Mean per cent (with standard deviation) of erroneous code on the skill-based, rule-based and knowledge-based level in the structural and behavioural part of the task when using plcML, modAT4rMS or FBD

| | % Skill-based errors | % Rule-based errors | % Knowledge-based errors | % Unclear | % Technical failure |
|---|---|---|---|---|---|
| plcML structural modelling | *M* = 0.25 | *M* = 21.50 | *M* = 0 | *M* = 12.50 | *M* = 0 |
| | SD = 0.71 | SD = 23.80 | SD = 0 | SD = 19.09 | SD = 0 |
| plcML behavioural modelling | *M* = 0.62 | *M* = 25.37 | *M* = 0.75 | *M* = 62.25 | *M* = 0 |
| | SD = 1.19 | SD = 31.10 | SD = 2.12 | SD = 26.49 | SD = 0 |
| modAT4rMS structural modelling | *M* = 0.67 | *M* = 6.37 | *M* = 0 | *M* = 0.33 | *M* = 0 |
| | SD = 1.74 | SD = 15.55 | SD = 0 | SD = 1.13 | SD = 0 |
| modAT4rMS behavioural modelling | *M* = 5.12 | *M* = 15.12 | *M* = 0.12 | *M* = 5.04 | *M* = 0 |
| | SD = 5.98 | SD = 21.90 | SD = 0.61 | SD = 7.14 | SD = 0 |
| FBD structural modelling | *M* = 0.85 | *M* = 14.85 | *M* = 0 | *M* = 7.75 | *M* = 0 |
| | SD = 2.18 | SD = 22.38 | SD = 0 | SD = 16.42 | SD = 0 |
| FBD behavioural modelling | *M* = 2.50 | *M* = 27.30 | *M* = 0 | *M* = 9.45 | *M* = 0.50 |
| | SD = 3.43 | SD = 33.92 | SD = 0 | SD = 20.77 | SD = 2.24 |

**Table 5** Summary of resulting statistics for all ANOVAs; significant effects are set in boldface

| Effect | *df* | Skill-based errors | | Rule-based errors | | Knowledge-based errors | | Unclear | | Technical failure | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *F* | *p* | *F* | *p* | *F* | *p* | *F* | *p* | *F* | *p* |
| Notation | 2, 49 | 3.00 | 0.059 | 1.61 | 0.210 | 2.03 | 0.143 | **19.91** | **<0.001** | 0.79 | 0.458 |
| Task (ST, BT) | 1, 49 | **8.59** | **0.005** | **6.96** | **0.011** | **4.31** | **0.043** | **89.39** | **<0.001** | 0.595 | 0.444 |
| Notation × task | 2, 49 | **3.09** | **0.055** | 0.53 | 0.593 | 2.03 | 0.143 | **45.07** | **<0.001** | 0.79 | 0.458 |

*ST* structural task, *BT* behavioural task

As far as skill-based errors are concerned, there is a significant interaction effect between notation and task, i.e. the two factors cannot be considered separately. A closer view on the data reveals that especially the behavioural modelling with modAT4rMS seems to be quite prone to errors of this kind.

Missed points that are assigned to the category "unclear", too, cannot be attributed to either notation or task effects but a combination of them. Attributable to lack of time, this phenomenon mainly occurs at behavioural modelling with plcML.

In Table 5, a summary of resulting statistics for all ANOVAs is given.

At the skill-based level, errors were caused by unintentional actions (e.g. mental or physical slip) during highly routine activities. Typical error types were as follows: forgotten elements (e.g. control signal, signal type, sensors for storing), errors due to an imprecisely studied task description, confused notation elements, typing errors and careless mistakes. As these errors do not always occur at the skill-based level, the subjects' statements during the modelling and in the interview afterwards were crucially important for the classification. A summary of all skill-based error types can be found in Table 6.

Errors at the rule-based level were caused by the intentional and consistent use of faulty rules that were mostly due to insufficient understanding of the notation's syntax or—in the case of OO modelling—due to problems with the rules of encapsulation, the creation of modules and the creation of variants and aggregations. An overview on rule-based error types is given in Table 7.

At the knowledge-based level, errors occurred only during behaviour modelling in plcML and in modAT4rMS, where a few of the participants had problems to fully understand the challenges of the task (0.75 % of all errors in plcML, 0.12 % of all errors in modAT4rMS).

Tables 6 and 7 provide a detailed analysis of errors explaining faulty modelling performances in different notations. The results help to understand the reasons and causes for deficits in structural and behavioural modelling. The analysis revealed specific deficits in concepts inherent to particular notations.

## 6 Discussion

Both the preliminary study of UML failures and the experiment with the error analysis indicated the "translation" problems from the task to an appropriate mental model as an important factor for modelling tasks as well as time. The mental model of the system and the application to the notation are essential for modelling tasks. Deficits in

**Table 6** Summary of skill-based error types in plcML, modAT4rMS and FBD (in %)

| | plcML | modAT4rMS | FBD |
|---|---|---|---|
| % Skill-based errors during structural modelling | | | |
| Forgotten control signal | 0.25 | 0.17 | 0 |
| Forgotten signal type | 0 | 0.08 | 0.30 |
| Forgotten sensors for storage belts | 0 | 0.33 | 0 |
| Task description studied imprecisely | 0 | 0 | 0.30 |
| Sensors partly created as bool | 0 | 0 | 0.15 |
| Confused notation element | 0 | 0 | 0.10 |
| % Skill-based errors during behavioural modelling | | | |
| Task description studied imprecisely | 0 | 1.67 | 0.95 |
| Typing error | 0.12 | 0 | 0.05 |
| Careless mistake == versus = versus := | 0 | 1.04 | 0 |
| Wrong sensor function | 0 | 0 | 0.45 |
| Confused notation element | 0 | 0 | 0.05 |
| "conveyor_done" attached to "sr.q" that starts the conveyor | 0 | 0 | 0.10 |
| Functions for signal forgotten to program | 0 | 0.08 | 0 |
| Forgotten timer | 0 | 0.37 | 0 |
| Forgotten reset | 0 | 0.46 | 0 |
| Forgotten SR | 0 | 0 | 0.20 |
| Forgotten task parts | 0 | 0.17 | 0.20 |
| Forgotten sensors for bottle storing | 0 | 0.33 | 0 |
| Forgotten signal at comparison | 0 | 0.21 | 0 |
| Forgotten type | 0 | 0.12 | 0 |
| Overall process at system level forgotten | 0 | 0.25 | 0 |
| Forgotten VOs | 0 | 0.17 | 0 |
| Forgotten dot operator | 0 | 0 | 0.20 |
| Transitions in method "Sortieren()" forgotten | 0 | 0.08 | 0 |
| Forgotten to switch-off the conveyor/storing belt | 0.50 | 0.08 | 0 |
| Behaviour of switch and conveyor belt forgotten | 0 | 0 | 0.30 |

**Table 7** Summary of rule-based error types in plcML, modAT4rMS and FBD

| | plcML | modAT4rMS | FBD |
|---|---|---|---|
| % Rule-based errors during structural modelling | | | |
| Problems with the rules of encapsulation and the creation of modules | 13.00 | 6.05 | 2.65 |
| Insufficient understanding of the creation of variants and aggregations | 8.50 | 0 | 0 |
| Insufficient rule knowledge of the notation's syntax | 0 | 0 | 10.80 |
| Insufficient understanding of the transfer of values | 0 | 0.33 | 0 |
| Insufficient understanding of the examination of the input state | 0 | 0 | 0.20 |
| Faulty function block call | 0 | 0 | 1.20 |
| % Rule-based errors during behavioural modelling | | | |
| Problems with the rules of method encapsulation | 20.62 | 9.79 | 8.60 |
| Insufficient understanding of encapsulation: module to (sub)module | 0 | 0.92 | 0 |
| Insufficient understanding of encapsulation: data access | 0 | 0.21 | 0.20 |
| Problems with the formation of modules and the rules of inheritance | 0.87 | 0 | 0 |
| Insufficient understanding of the difference between function and operation | 0 | 0.17 | 0 |
| Insufficient understanding of cross-module transfer of values | 0 | 1.33 | 0 |
| Insufficient rule knowledge of the notation's syntax | 0.37 | 0.33 | 16.55 |
| Insufficient understanding of network sequence (in FBD) | 0 | 0 | 0.70 |
| Insufficient understanding of operators | 0 | 1.96 | 0.15 |
| Insufficient understanding of the timer function | 1.50 | 0.58 | 0.90 |
| Insufficient understanding of the difference between `transition` and `state` (in plcML) | 1.37 | 0 | 0 |
| Insufficient understanding of task sequence: no return to `state` but new `state` required | 0.12 | 0 | 0 |
| Insufficient understanding of transition conditions | 0.50 | 0 | 0 |

the translation of the mental model in the modelling system were the main reasons for the shortcomings in the performances. The error analyses revealed predominantly different rule-based errors in behaviour and structure modelling performance. For instance, the rules of encapsulation and the creation of the modules were deficient. The lack of time for modelling tasks performance was a problem in both studies which is a result of less prior experience and practice. It should be noted that results of error analysis may be different with more experienced participants with more competence in modelling tasks.

Unintentional errors at the skilled-based level were forgotten elements, errors due to an imprecisely studied task description or confused notation elements. Skilled-based errors are due to inattention or memory problems, and this could partly be caused by the think-aloud technique but also less practice with the modelling tasks. The classification as rule based also reflects that those kinds of tasks need more practice and experience to develop and apply rule knowledge as required for successful task performance. Rule-based errors result from intentional and consistent use of faulty rules. The 2-day training is only a short period of time to develop abilities for modelling tasks. Although the test task was very similar to the training task, the transfer of new learned rules and knowledge and their application in the new task was limited and results in the use of faulty rules. Only an incomplete mental model could be developed during the training session. As a result of the error analysis, we further gained knowledge about the rules and concepts that are error prone, and therefore, cause errors in modelling. In contrast to prior results (e.g. Vogel-Heuser et al. 2012, 2013), the detailed analysis of errors in modelling task is attributed to concepts and characteristics inherent to different notations.

Comparison of the different notations (plcUML, modA4rMS, FBD) showed that behavioural modelling with modA4rMS is quite more error prone than the other notations. The modA4rMS led to more unintended failures due to inattention (e.g. forgotten reset, forgotten timer) and careless mistakes. There were no significant differences in the occurrence of rule-based errors. Knowledge-based errors rarely occurred. During behaviour modelling in plcML and in modA4rMS, comprehension problems about the challenges of the task appeared.

As already outlined in the introductory part of this paper, the taxonomy presented above has two main application areas: education and tool design.

### 6.1 Implications for education

The error analysis by the use of SRK model in mechanical engineering provides a detailed description about concepts and requirements of notations which cause errors in structural and behavioural modelling tasks. On the one hand, it can be helpful to get to know the errors that mechatronics apprentices and technicians commonly make in FBD and in which respects these differ to those made with plcML and with modAT4rMS. Hereafter, the knowledge about which errors occur when and why can be used for improving education, as students can deliberately be advised, which errors are mainly to be expected and how they can be detected and resolved successfully. Moreover, instructors have the possibility to become more alert about common errors and misconceptions and can specifically address aspects of the notation to be learned that are generally poorly received by their apprentices. The awareness of difficulties in learning rules or the requirements of elements provide could further be used for the development of training or tutorials in education.

In the study described above, knowledge-based errors only had a marginal effect on the total number errors (independent of the notation used). This in part can be attributed to the previously completed training, which included a very similar task. Secondly, it is very likely also due to the simplicity of the task, which was relatively undemanding in terms of logic and problem solving skills.

In all three notations by far the most common were rule-based errors, especially during the behavioural modelling part of the task. So, it is in particular on this level, where teachers and students should act, if they want to achieve a reduction of programming errors. With FBD, instructors should place more emphasis on teaching the notation's syntax rules. In contrast, plcML and modAT4rMS seem to be more prone to errors that are due to misinterpreted rules of OO fundamentals. Here, a training focusing on encapsulation rules, the formation of modules and the formation of variants and aggregations are most promising.

Skill-based errors were only a minor problem in our study, but are likely to become more relevant once the apprentices are familiar with the notation's rules. On this level, most errors occurred in modAT4rMS—a notation that produces significant less mental workload than the other two languages (Braun 2013). As these errors typically were only small slips or lapses, participants tended to overlook them and to focus on less familiar parts of the code. Given the fact that the correct solution is known to them, tuition presumably is of limited effectiveness here.

Outstanding regarding plcML was the fact that the time required for structural modelling often was enormous, often leading to a hardly processed behaviour modelling part. Although there were few errors in the strict sense, it seems obvious that the apprentices' solutions were quite inefficient. A training focusing on this aspect might improve the overall quality of the created models.

### 6.2 Suggestions for tool design

On the other hand, the taxonomy provides inspiration for the future design of tools that support apprentices and technicians at detecting and fixing errors within their code.

The results of the presented study make it advisable to choose different feedback approaches, depending on the level where the error occurred in order to be most effective. As skill-based errors are unintentional and not based on faulty knowledge, it is likely to be sufficient to indicate their existence and their position. A check similar to a front-end compiler analysis that validates lexical correctness, syntax and semantics (e.g. type checking and object binding) and issues warnings (if necessary) already may help counter these errors.

As rule-based errors are concerned, such a tool could be instrumental, too. However, it is likely that only common syntax and semantic errors can be identified by these comparably simple heuristic checks. For instance, a class that is used only once is likely to be a mistake and the programmer can be drawn to this fact—possibly by offering additional warning information and (short) explanations in the event that an error is caused by insufficient rule knowledge.

At all levels, a virtual testing environment could provide further feedback that is likely to help programmers at detecting and resolving errors. Nevertheless, this should never be the only mode of feedback: for example, code often compiles, although it may still contain plenty of bugs. If there is no time for extended testing, these errors remain unresolved and may reinforce the programmer's faulty rule knowledge—making it even more difficult to be corrected.

Another promising approach could be the further development and optimization of the modelling notations themselves, and in particular, of the modAT4rMS notation that has proved to be the easiest and most user friendly of the three tested notations (Braun 2013). However, problems with the rules of encapsulation and the creation of modules were still the most prominent error sources with modAT4rMS. Consequently, the greatest potential for a further reduction of the number of errors and thus an increase of the overall modelling performance seems to lie in a better support of the user during encapsulation and the creation of modules. If this should be done through an adaptation of the modelling language itself or through a modification of the modelling procedure (or both), this needs to be clarified in subsequent studies.

### 6.3 Evaluation of the method

The described methodology of error analyses in modelling tasks has proved its usefulness for studying apprentices' failures when writing code in plcML, modAT4rMS or FBD and the differences between them. However, it also has some limitations: first of all, it requires experience with human subjects and the think-aloud paradigm, with the skill-rule-knowledge framework and with coding according to grounded theory. Second, it also requires extensive knowledge in all three modelling languages that are compared in order to be able to identify the errors that were made by the subjects as well as being able to ask questions in the follow-up interview that are helpful for the subsequent analysis by the experimenters. Another limitation is the time required for data collection and analysis, which may take weeks (or even months).

However, these trade-offs have to be weighed against the benefits of the method described: we gained far more specific and practically usable information as it would have been possible with more economic methods like task analysis. Moreover, direct observation combined with the think-aloud technique led to deeper insights than it would be possible with the previously applied methods of code inspection and/or interviews and made the interpretation and classification of the occurred errors much simpler. Nevertheless, the assignment still was anything but easy. For the participants, the think-aloud technique was quite an unfamiliar task. From time to time, the experimenter had to encourage the participants to verbalize their thought process. The method also has several constraints, for instance, the unaccustomed verbalization of thoughts during task performance. Thinking aloud also affects cognitive processes which results in time delay for task performance or changes of behaviour. The lack of time was also a problem in the described experiment.

The coding process by the use of open and axial coding from grounded theory (Strauss and Corbin 1998) is an iterative process. Analysts considered collected data several times. At first noted occurrence of an error, afterwards, a coding scheme and labels for chunks are developed, and finally, categories of the classification were refined and/or combined to umbrella terms. The analysts were a team of software experts and a psychologist that examined the data in an iterative process. The method of the grounded theory did not analyse previously defined categories, but rather generate the coding scheme and categories in the analysis of the data. This implies the constant comparison of data according to similarities or differences. Finally, a classification results with distinct categories derived from the data. In particular, the distinction between rule-based and knowledge-based errors often led to lengthy discussions between the analysts. It was decided to classify faulty code that is obviously due to incomplete rule knowledge as being rule based, as the subjects had successfully applied the necessary knowledge during the training 2 days before in a very similar task that was even more difficult. However, it should be kept in mind that the differentiation between these two categories is quite delicate. According to the method applied for analysis, the reliability could not be proved as other methods with previously defined classification provide. Therefore, the explanatory power of the results is limited. Nevertheless, we could show a comprehensive analysis of errors and by the use of the SRK model errors are attributed to the different causes. The knowledge gained has several implications of education and tool design as mentioned above.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

## References

Anderson J, Jeffries R (1985) Novice LISP errors: undetected losses of information from working memory. Hum Comput Interact 1(2):107–131

Blackwell AF (2002) First steps in programming: a rationale for attention investment models. In: Proceedings of the IEEE symposia on human centric computing languages and environments, pp 2–10

Boren MT, Ramey J (2000) Thinking aloud: reconciling theory and practice. IEEE Trans Prof Commun 43(3):261–278

Braun S (2013) Adaptation and practical evaluation of notations for modular, object-oriented programming of open loop control systems in machinery and plant engineering in the context of usability. Ph.D. thesis at Mechanical Engineering Department, TU Munich

Davies SP (1994) Knowledge restructuring and the acquisition of programming expertise. Int J Hum Comput Stud 40(4):703–726

Eisenstadt M (1993) Tales of debugging from the front lines. In: Empirical studies of programmers, 5th Workshop, Palo Alto, CA, pp 86–112

Ericsson KA, Simon HA (1984) Protocol analysis: verbal reports as data. MIT Press, Cambridge, MA

Hajarnavis V, Young K (2008) An assessment of PLC software structure suitability for the support of flexible manufacturing processes. IEEE Trans Autom Sci Eng 5(4):641–650

International Electrotechnical Commission (2003) IEC International standard IEC 61131-3: programmable logic controllers, part 3: programming languages

Ko A, Myers B (2005) A framework and methodology for studying the causes of software errors in programming systems. J Vis Lang Comput 16:41–84

Mayrhauser Av, Vans AM (1997) Program understanding behaviour during debugging of large scale software. In: Empirical studies of programmers, 7th workshop, Alexandria, VA, pp 157–179

Panko R (1998) What we know about spreadsheet errors. J End User Comput 10(2):5–21

Panko RR, Sprague RHJ (1999) Hitting the wall: errors in developing and code-inspecting a 'simple' spreadsheet model. Decis Support Syst 22:337–353

Park TH, Saxena A, Jagannath S, Wiedenbeck S, Forte A (2013) Towards a taxonomy of errors in HTML and CSS. In: Proceedings of the 9th annual international ACM conference on international computing education research. ACM, pp 75–82

Perkins DN, Martin F (1986) Fragile knowledge and neglected strategies in novice programmers. In: Empirical studies of programmers, 1st workshop, Washington, DC, pp 213–229

Purao S, Rossi M, Bush A (2002) Towards an understanding of the use of problem and design spaces during object-oriented system development. Inf Organ 12:249–281

Rasmussen J (1983) Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. IEEE Trans Syst Man Cybern 13(3):257–266

Reason J (1990) Human error. Cambridge University Press, Cambridge

Robins A, Rountree J, Rountree N (2003) Learning and teaching programming: a review and discussion. Comput Sci Educ 13(2):137–172

Shackelford RL, Badre AN (1993) Why can't smart students solve simple programming problems? Int J Man Mach Stud 38:985–997

Siau K, Loo P (2006) Identifying difficulties in learning UML. Inf Syst Manag 23:43–51

Siau K, Tian Y (2001) The complexity of unified modelling language: a GOMS analysis. In: 14th international conference on information systems, New Orleans, December 16–19, pp 443–448

Spohrer JG, Soloway E (1986) Analyzing the high frequency bugs in novice programs. In: Empirical studies of programmers, 1st workshop, Washington, DC, pp 230–251

Strauss A, Corbin J (1998) Basics of qualitative research: techniques and procedures for developing grounded theory. Sage Publications, Thousand Oaks

Thramboulidis K, Frey G (2011) Towards a model-driven IEC 61131-based development process in industrial automation. J Softw Eng Appl 04(04):217–226

Vogel-Heuser B, Braun S, Obermeier M, Jobst F, Schweizer K (2012) Usability evaluation on teaching and applying model-driven object-oriented approaches for PLC software. ACC, Montréal

Vogel-Heuser B, Obermeier M, Braun S, Sommer K, Jobst F, Schweizer K (2013) Evaluation of a UML-based versus an IEC 61131-3-based software engineering approach for teaching PLC programming. IEEE Trans Educ 56(3):329–336

Vyatkin V (2013) Software engineering in factory and energy automation: state of the art review. IEEE Trans Ind Inf 9(3):1234–1249

Witsch D, Vogel-Heuser B (2011) PLC-statecharts: an approach to integrate UML-statecharts in open-loop control engineering: aspects on behavioural semantics and model-checking. In: 18th IFAC World Congress, Milano

Youngs E (1974) Human errors in programming. Int J Man Mach Stud 6:361–376

Zoitl A, Vyatkin V (2009) IEC 61499 architecture for distributed automation: the 'Glass Half Full. IEEE Ind Electron Mag 3(4):7–23