Teaching Agile Model-Driven Engineering for Cyber-Physical Systems

Jan Oliver Ringert[1,2], Bernhard Rumpe[2], Christoph Schulze[2], Andreas Wortmann[2]
[1]School of Computer Science, Tel Aviv University, Tel Aviv, Israel, http://cs.tau.ac.il
[2]Software Engineering, RWTH Aachen, Aachen, Germany, http://www.se-rwth.de/

***Abstract*—Agile development methods, model-driven engineering, and cyber-physical systems are important topics in software engineering education. It is not obvious how to teach their combination while respecting individual challenges posed to students and educators. We have devised a software project class for teaching the agile MDE for CPS. The project class was held in three different semesters. In this paper, we report on the setup of our exploratory study and its goals for teaching. We base our evaluation and insights on interviews and questionnaires. Our results show the feasibility of combination of agile MDE for CPS but also the challenges this combination poses to students and educators.**

***Keywords*-teaching; model-driven engineering; cyber-physical systems; case study;**

I. Introduction

Software development projects represent an integral part of most computer science and software engineering (SE) curricula. Typically, students develop non-trivial pieces of software in teams over extended periods of time, *e.g.,* the duration of a semester. The teaching goals of these project classes are usually to expose students to real-world software development tasks using recent technologies and techniques to demonstrate practical but also social challenges for developing software in teams.

Previous works reported on the benefits and challenges of teaching SE in project classes in various settings [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. Important parameters of a software development project are the pursued development process, the employed development techniques, and the nature of the developed system. Agile software development [11] refers to development processes built on principles of collaboration, self-organization, and fast response to change rather than strictly defined and prescribed tasks. Model-driven engineering [12] (MDE) puts software models in the focus of system engineering. It is a prevalent technique in important application domains [13], [14], [15], [16]. There are however still challenges for the complete adoption of MDE [14], [17]. Cyber-physical systems [18] (CPS) is a term to describe systems in the intersection of computational (cyber) and physical processes. Such systems, with all their challenges to software engineers (the dynamics of their physical part are not easily controlled), are of great importance for toady's industry and academic research. Popular examples of CPS are automated mobile systems, smart grids, and robotics systems.

The combination of agile development with MDE fits well with teaching goals in the CS curriculum: to develop software in a team and learn about collaboration challenges. In addition, the application area of CPS appears well-suited to attract and motivate students. Previous works have reported on employing CPS as beneficial to the teaching and learning experience [1], [2], [3]. There is, however, a lack of reporting on well-designed curricula for teaching MDE [5], [6], [19] and on software project classes in general [9].

In this paper is we report results from case studies about teaching agile MDE of CPS in software project classes. We consider the following research questions in this context:

- Which challenges does MDE pose over using GPLs?
- Which development challenges result from CPS?
- Which Scrum elements are perceived most beneficial?
- What should guide MDE and CPS technological choices?

The case studies we have conducted address these questions in an exploratory manner. To this effect, we employed a variant of Scrum for MDE with robotics platforms in the project classes. We have executed the case study in three consecutive years with teams of different students.

Objective The objective is to explore challenges and benefits of teaching agile SE project classes using MDE and CPS.

Case The case we investigate is a one-semester software project class for undergraduate students (executed in iterations over three years)

Theory Our case study employs agile practices of Scrum [20], the MDE tools of MontiArcAutomaton [21], and LEGO Mindstorms NXT[1] and Festo Robotino[2] robots as CPS.

Methods We used two kinds of data collection methods: interviews and notes during Scrum meetings and questionnaires filled out by students.

Selection Strategy The selection of participating students was handled centralized by the CS faculty and the selection of the case under the control of the authors.

After each iteration of the case study, we have adjusted the next iteration based on lessons learned. The students of each iteration were presented different scenarios and technologies for MDE of CPS. We report on the results of all three case studies and discuss the differences and commonalities of their execution as well as of the collected data.

Our studies show that a combination of MDE and CPS can effectively be taught in project classes. All teams managed

[1]LEGO Mindstorms NXT: http://mindstorms.lego.com/
[2]Robotino website: http://www.festo-didactic.com/int-en/learning-systems/education-and-research-robots-robotino/

978-1-5386-2671-9/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-SEET.2017.16

to produce working CPS. We learned that (1) stability and robustness of functional capabilities are major criteria for selecting teaching CPS platforms; (2) MDE requires additional teaching effort; (3) Scrum must be adjusted to student needs; and (4) employed technologies must align with teaching goals.

We detail the subject of our research further in Sec. III and present our study design in Sec. IV. An overview of the three independent project classes is given in Sec. V. We provide observations in Sec. VI and a discussion in Sec. VII. Finally, Sec. VIII concludes.

II. Related Case Studies

Our approach to teaching a realistic SE experience relies on agile methods, model-driven engineering, and their application to cyber-physical systems. This section presents related case studies along these three dimensions summarized in Table I.

TABLE I
CASE STUDIES TEACHING AGILE DEVELOPMENT AND/OR MODEL-DRIVEN ENGINEERING AND/OR DEVELOPMENT FOR CPS.

Source	Agile	MDE	CPS	Description
[1]	–	–	✓	ANSI-C education w. RCX robots
[2]	–	–	✓	Java introduction w. RCX robots
[3]	–	–	✓	Teaching A.I. w. RCX robots
[4]	–	✓	✓	DSL engineering for mobile robot
[5]	?	✓	–	MDE of a communication system
[6]	–	✓	–	MDA and transformations
[8]	–	✓	–	Modeling CRUD applications
[7]	✓	✓	✓	Agile MDE with NXT robots
[9]	✓	✓	?	Multiple projects w. ext. customers
[10]	✓	–	–	Teaching agile in high school

Various studies have employed LEGO Mindstorms robots in computer science education. The authors of [1] report on using LEGO Mindstorms RCX robots (the predecessor of NXT) to teach programming with ANSI-C to 80 undergraduate freshmen. The freshmen build and programmed six RCX robots for educational examples (such as drawing on the floor or escaping a maze). The author of [3] reports on the results from teaching artificial intelligence with LEGO RCX robots and an ANSI-C middleware in two classes to a total of 21 students. Both case studies neither focused on teaching MDE, nor on agile development. In [2], RCX robots were investigated as vehicle for teaching introductory Java. In a similar vein, the authors of [22] propose a roadmap for teaching Java with RCX robots. Both investigations do not perform case studies.

The author of [4] reports on a project class teaching MDE with a robotics platform based on the Parallax Stamp[3] robot in a five week intensive class to two groups of 5-6 students each. The class focused on DSL and code generator development and the author does not report on the impact of CPS to teaching or the usage of agile methods.

The organizers of a class on teaching software development with LEGO NXT robots and the LeJOS middleware performed a four question survey on the usefulness of using LEGO NXT robots for teaching [23]. The results indicate that using these robots is motivating and useful for the students. They did neither employ agile methods, nor MDE.

The authors of [5] found that "teaching students how to use models effectively" is a prime concern in SE education. They present a survey from integrating MDE into a software design class. In this class, 14 students employed a modeling language to generate a (purely software) communication system. Whether the student teams employed agile processes is not documented. The authors of [9] identify designing project classes "realistic enough" as one of the main challenges in teaching software engineering. They employ an agile development process called "Rugby", which is a variant of Scrum tailored to student requirements (identified as "part-time developers"). Similar to our approach, Rugby is based on weekly instead of daily meetings and focuses on continuous delivery. The authors report on four project classes with over 300 students working for various external clients. These classes required the students to use UML for informal modeling only. Data was collected via an online survey and its results indicate that Rugby is considered beneficial by the students and that the students improved their programming skills as well as their modeling skills. The authors of [19] similarly identify teaching realistic MDE as the "main challenge we face now is to succeed in building specialized high-level curricula so that engineers will be able to rapidly deploy MDE solutions in industry". The authors of [6] report qualitative experiences from applying MDA with a group of 52 students in a 13 week project class. In this class, the students faced challenges in DSL development and model transformation. The main findings are that (1) teaching MDA cannot reuse previous knowledge from code-centric approaches; (2) there is a lack of efficient tools; and (3) there is no "good textbook" yet. The authors of [24] report on a teaching metamodelling and MDE tooling development using UML to up to 150 master students. A case study on teaching 'pragmatic' MDE [8] to a group of 58 students by modeling CRUD applications and generating code for these also performed a survey at the end. This survey of 10 questions hints at the positive effect of a pragmatic approach to teaching MDE. None of these studies focused on agile development for CPS.

A two semester case study on teaching agile methods [25] with a total of 88 students identified the challenge that teaching agile development in realistic projects tends to shift focus to technical issues instead of process issues. To mitigate this, the authors propose various changes to their classes including agile mini projects in the beginning of the class and removing software requirements from student assessment. Instead, assessment should be solely focused on the students' use of agile practices. The classes neither did cover MDE, nor CPS, but focused on agile methods. A study with four high school classes [10] employed a survey to investigate the student reception of agile development during a case study focusing on pure software development. The authors found that the students neither enjoyed employing test-first development, nor user stories nor time boxing. The effect of pair programming, however, was considered beneficial by the

[3]Parallax Stamp website: https://www.parallax.com/robots/robots-overview

students.

Finally, we have reported preliminary results of the first iteration of the project class in [7]. Our current work extends this report with data and findings from two additional years employing industrial robots instead of LEGO NXT robots and more advanced software platforms. In addition, we have included a focus on Scrum in the latter two iterations of the class.

III. Research Context

This section gives a brief overview of the research subject by providing background on the techniques employed in our case studies. This includes our choices for an agile development method, MDE, and CPS. It also describes the context of the class in the computer science curriculum.

1) Agile: Agile software development [11] refers to development processes built on principles of collaboration, self-organization, and fast response to change rather than strictly defined and prescribed tasks. Requirements are often informally documented in user stories and refined, implemented, and validated together with stake holders during the project. A popular process model for agile development is Scrum [20]. Scrum consists of iterative sprints – time boxed development phases – which contain daily Scrum meetings. Each sprint begins with a sprint planning meeting in which the work for this sprint is defined (and documented in the sprint backlog). It ends with a sprint review on the product and a sprint retrospective on the process. Ideally, each sprint produces a working deliverable that improves customer value. Major roles in the process are (1) the product owner, who represents the customers and prioritizes backlog items accordingly; (2) the Scrum master, who is responsible for Scrum process quality; and (3) the development team.

2) MDE: Model-driven engineering [12] puts software models in the focus of system engineering. It is a prevalent technique in important application domains [13], [14], [15], [16]. MDE uses models as main development artifacts that serve communication as well as code generation. Development is model-driven but integration with existing or newly developed code is often required. One framework for the MDE of robotic systems is MontiArcAutomaton [21]. It combines MDE of a system's architecture and its behavior with powerful extension mechanisms and integration of handcrafted code and frameworks. The main modeling formalisms employed in our case study are component and connector models [26] and automata to describe reactive behavior [27]. MontiArcAutomaton supports code generation to multiple platforms [7], [28].

3) CPS: Cyber-physical systems [18] (CPS) is a term to describe systems in the intersection of computational (cyber) and physical processes. Such systems, with all their challenges to software engineers (the dynamics of their physical part are not easily controlled), are of great importance for toady's industry and academic research. We have selected the educational robotics framework LEGO Mindstorms NXT for the first class to demonstrate CPS development. Code run on the robots was either generated from MontiArcAutomaton or written in Java using the LeJOS[4] API. The second and third iteration of the class employed the industrial Robotino platform using again MontiArcAutomaton code generators and extensions developed for ROS [29] in Python and for SmartSoft [30].

4) Curriculum: The project class is implemented as a mandatory module in the CS bachelor and international master curriculum. It is taught by the whole CS faculty. The different groups compete for the best students which compete for limited project places. Some prescribed goals of teaching are:

- Using state-of the art/modern tools.
- Develop simple software architectures.
- Test software.
- Self-management.
- Group effects and team challenges.
- Presentation of results.

Students of the CS bachelor program had an introduction to Java in a programming class and to MDE in a general SE class. Students of the international master degree were offered an optional class on MDE with focus on UML. At the time of our case studies, neither MontiArcAutomaton, nor general component & connector (C&C) architecture modeling were taught in any class.

IV. Study Design

The main focus of our case studies is the exploration of teaching the agile model-driven engineering of cyber-physical systems as a project class. We designed case studies around the following guiding questions of our exploratory study:

R1 Which challenges does MDE pose over using GPLs?
R2 Which development challenges result from CPS?
R3 Which Scrum elements are perceived most beneficial?
R4 What should guide MDE and CPS technological choices?

The case studies were executed as classes for the duration of one semester. They were taught in winter $2012/13$ (in the following denoted by $C1$), winter $2013/14$ ($C2$), and summer 2014 ($C3$) at RWTH Aachen University by two supervisors each. All three case studies used the MontiArcAutomaton [21] modeling language and tools in combination with different GPLs. This allows us to address question **R1**. The case studies cannot be seen in a comparative manner or as replications. In each execution, the target CPS and the employed technologies changed. We present these differences in Sec. V. Employing Scrum in all three case studies allows us to address question **R3** and the different choices of CPS allow a broader view on questions **R2** and **R4**. We now describe the structure of the class and data collection common to all case studies.

A. Class Structure and Subjects

Project classes are mandatory for students at RWTH Aachen University and are awarded with 7 ECTS for successful participation (16 weeks). Classes are centrally organized for students from various degree programs, including computer science bachelor and master, software systems engineering, and media informatics. The central organization influenced

[4]LeJOS website: http://www.lejos.org

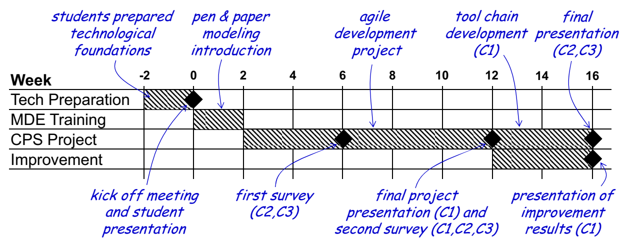

Fig. 1. Project class structure: Two weeks before semester begin, the students prepared technological foundations, which were presented at the kickoff meeting. After two subsequent weeks of modeling training, the CPS project started. Only on $C1$ this was followed by a tool chain improvement part. In $C2$ and $C3$ the last four weeks were part of the CPS project.

our case design in various ways. First, we had to make sure the class respects globally prescribed teaching goals (see Sec. III-4). Second, the duration of the case study was limited. Finally, our selection of participants was regulated. However, these *restrictions* in case design fit naturally to our objective of analyzing this kind of class.

We selected between 8 and 14 students according to their preferences and motivation statements each year. The students had different levels of experience with MDE (see Sec. III-4). Java is the default GPL for many classes at RWTH Aachen University and taught in the first semester of the computer science bachelor. Thus, most students had experience in programming with it. As none of the advanced MDE lectures is mandatory, modeling experience ranged from none to modeling exercises with UML. Most of the students neither had experience in agile development, nor in SE for CPS.

All classes comprised at least one weekly meeting of two hours. Whenever problems occurred, additional meetings were scheduled. Students employed a variant of the Scrum methodology. Similar to [9], we changed the daily Scrum meetings, which were infeasible due to the students' varying schedules, to weekly meetings. The meetings also included student presentations on the progress of their tasks. Due to the small team sizes, the Scrum masters also acted as developers. Finding that enacting both roles is more challenging than expected, we changed this to employing teams of two ($C2$) and three ($C3$) Scrum masters. Students were evaluated based on their progress reports, development activity, and final results.

We structured the project classes into multiple stages as depicted in Fig. 1: two weeks prior to semester begin, each student was assigned a topic relevant to the class and we provided introduction materials (publications, websites, tutorials) to prepare that topic. The topics included MontiArcAutomaton [21], its underlying language workbench MontiCore [31], Scrum, the robotics middleware to be employed, and the development infrastructure (*e.g.,* Maven, continuous integration, *etc.*). The most important topics were assigned to multiple students to ensure sufficient expertise in the teams. The complete list of topics is available in the appendix. After two weeks of preparation, the students presented the technologies to their colleagues and the supervisors at the kickoff meeting. Afterwards, the supervisors presented the user stories to the project class

To become familiar with MontiArcAutomaton and its behavior modeling capabilities, the students performed small modeling tasks using pen and paper (*e.g.,* designed initial system architectures and behavior models) in the subsequent two weeks. Afterwards, the team elected Scrum master(s), split into development teams along the system functionalities, components and related requirements identified in the paper modeling stages. Then they began the respective robotics logistics projects. During that stage, the two supervisors enacted the roles of customers. Development ended with a final presentation of the working robotic systems.

In $C1$, which used a less complex robotics platform, the last four weeks were reserved for MontiArcAutomaton infrastructure improvement. This stage was dropped from the other project classes, due to negative feedback.

B. Data Collection

We employed two forms of data collection throughout all case studies. We conducted interviews with all students and asked them to fill out questionnaires

1) Interviews: We conducted informal interviews with the students on a weekly basis. These took part in open meetings after daily Scrum and sprint planning. In the interviews, we mainly addressed MDE (**R1**) and CPS (**R2**) challenges. In early meetings and before deadlines, when students struggled with organizational challenges, also Scrum practices (**R3**) were discussed. It is important to note that we treated these interviews also as part of our teaching activities, *i.e.,* the interview meeting was also used for giving advice and trying to actively address challenges the students faced. The result of interviews are informal personal notes summarized in Sec. VI.

2) Questionnaire: We have used a questionnaire to systematically capture the opinion of students about their project. The questionnaire addresses aspects of our guiding questions. We handed out questionnaires at the end of the development stage. In the second and third project class, we also conducted a first survey after four weeks of development.

The questionnaires are composed from sections investigating the students' learning effort, model usage, development effort, and process perception. The last section about Scrum was not part of the questionnaire in the first case study. The questionnaires comprise five types of questions inquiring (1) percentage values; (2) Likert scales from 1 to 10; (3) frequencies; (4) yes/no/don't know tuples; and (5) ordering of the answer options. The complete questionnaires' sections – with aggregated results – are presented in Sec. VI.

a) Learning Effort: The first six questions of the questionnaires inquire how long the students spent on understanding the respective project classes' technologies, applying these, making conceptual mistakes, and the effort in understanding their team members' artifacts. The questions aim to uncover challenges of using MDE over GPLs (**R1**), hint at the development challenges arising from CPS (**R2**), and may guide the selection of technological choices (**R4**). The repetition of the questions (after 4 weeks and end of development) intends to examine the effect of teaching MDE over time.

b) Model Usage: The second section of the questionnaires includes ten questions on the usage and reception of artifacts. It queries the students' confidence in their artifacts and their team members' artifacts, the effort to fix bugs in these artifacts, and their usage in terms of documentation, testing, and reuse. The questions of this section aim at uncovering how the students' familiarity with the modeling languages developed over time and whether the the central modeling artifacts are sufficiently comprehensible. Questions 13-14 examine whether the students consider behavior models suitable to CPS development and hints at whether this combination of modeling and domain challenges is suitable for further project classes (**R4**). Questions 15-16 aim at uncovering the complexity of development technologies with respect to each other (**R2**, not in first case study). Again, this section allows a comparison of MDE vs. GPL (**R1**), but also an assessment of all technologies employed (**R2**, **R4**).

c) Agile Process Reception: The last section of the questionnaire was part of the second survey of the latter two classes only and contains three questions on the the applicability of our Scrum variant and its implementation artifacts (**R3**).

V. The Project Classes

The individual project classes aim at introducing students to agile MDE with architecture description languages in the context of complex systems to resemble and prepare for state-of-the-art industry practice (as advocated for, *e.g.,* in [16], [9]). The teaching objectives are that the students (1) master agile software engineering using a variant of Scrum and its activities and artifacts; (2) learn model-driven engineering with architecture description languages; (3) familiarize with modeling infrastructure including model checking and code generation; (4) solve a 'realistic' project that requires models as primary development artifacts;

A. NXT Java Coffee Delivery ($C1$)

In the project class of winter term 2012/13, eight computer science master level students used the MontiArcAutomaton ADL with embedded automata to develop a robotic coffee delivery system as reported in [7]. The system[5], depicted in Fig. 2, consists of three LEGO NXT robots and uses the LeJOS NXT Java middleware to interface the robots. Consequently, the students used MontiArcAutomaton's Java code generator and implemented behavior of technical components in Java where necessary.

The system enables users to request coffee via a website hosted on a smart phone server. The smart phone is connected to the coffee preparation robot shown in Fig. 2 via Bluetooth. Upon receiving a request, the server commands the coffee delivery robot to load a plastic mug provided by the mug provider robot. The coffee delivery robot then drives to the coffee preparation robot, signals it via Bluetooth to

[5]Winter 2012/13 video: https://www.youtube.com/watch?v=xvlYN-6awfk

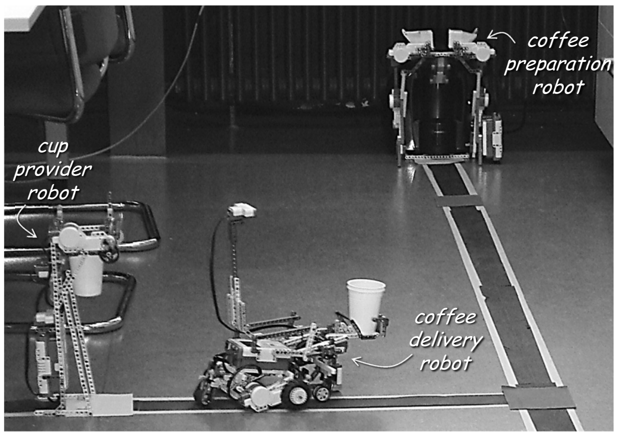

Fig. 2. The distributed robotic coffee service implemented with LEGO NXT robots using the LeJOS JVM as middleware.

prepare coffee, and finally drives to the user who requested the coffee. Lacking sophisticated localization, the students created a navigation system based on black lanes with colored junctions, where the sequence of colors uniquely identifies a room. The software architecture of this system comprises 35 component type definitions (23 specific to the project and 12 imported from provided libraries) in 60 component instances. Of the component types, 15 are composed and 8 are atomic. All 8 atomic components yield behavior automata and the 12 imported atomic component types had handcrafted Java implementations.

B. Robotino ROS Python Transport Service ($C2$)

Inspired by the iserveU service robotics research project [32], we updated the project class to feature a more realistic robotic logistics scenario using a Festo Robotino[6] robot. In the winter term of 2013/14, nine computer science master level students were assigned the task to develop a robotic logistics application that supports hospital staff (*i.e.,* pick up and deliver items, guide and follow persons).[7] The students again used MontiArcAutomaton with automata, but with Python code generation for the robot operating system (ROS) [29] to interface the Robotino robot. Hence, they also implemented component behavior in Python where necessary. While there is an, at the time, unstable Java implementation of ROS[8], we switched to Python to mitigate the effect of the students' previous knowledge of Java.

The robot, depicted in Fig. 3, uses a front-mounted laser sensor and integrated distance sensors to navigate and a Kinect and speakers for user interaction. The software architecture deployed to solve the logistics challenges with this robot comprised 31 component types used in 39 instances. 5 of the component types are composed and 26 are atomic. Of the atomic component types, 9 contain embedded behavior

[6]Robotino website: http://servicerobotics.eu/robotino/
[7]Winter 2013/14 video: https://www.youtube.com/watch?v=u6LF8KjvDgM
[8]ROSjava website: http://wiki.ros.org/rosjava

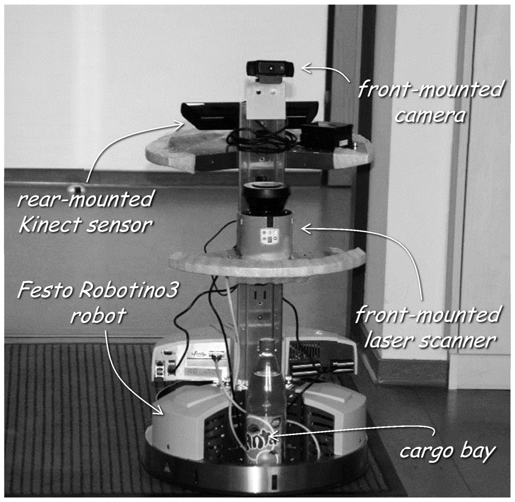

Fig. 3. The Robotino ROS transport service robot with Kinect, speakers, and front-mounted laser scanner.

models and the remaining 17 atomic component types rely on handcrafted Python implementations.

C. Robotino SmartSoft Java Transport Service ($C3$)

In summer term 2014, we assigned the task of developing a robotic logistics application to 14 students from different computer science programs.[9] To interface the Robotino, a tablet PC and a website were used. In this class, the students controlled the robot via the SmartSoft [30] middleware. SmartSoft was controlled via a protocol for which we provided a Java implementation. The students employed MontiArcAutomaton with extended automata that support Java in guards and actions. This extension increases expressiveness of behavior models but students also used Java for providing component behaviors not conveniently expressible with automata. The resulting software architecture comprises 28 component types, of which 6 are composed. Of the atomic component types 4 contain behavior automata and 18 have Java implementations. In total, the architecture comprises 32 component instances.

VI. Observations and Results

This section describes the observations from the questionnaires, and the interviews. Tables I-III present the survey questions with aggregated (average of provided answers) results (1) from the single questionnaire performed in $C1$ as $C1_2$; (2) from the first and second questionnaire of $C2$ as $C2_1$ and $C2_2$; and (3) from the first and second questionnaire of $C3$ as $C3_1$ and $C3_2$. We discuss our observations along the four research questions and lessons learned throughout the three case studies.

R1: Which challenges does MDE pose over using GPLs?

Teaching: It is important to note that students were familiar with the employed GPLs or at least their foundations from previous classes. However, the concepts of MDE and MontiArcAutomaton modeling languages were new to most students. This strongly influenced the plan of the project class by reserving additional time for technology seminars and architecture modeling feedback (see Sec. IV-A). Understanding and working with the synchronous message passing semantics of the behavior modeling languages (based on Focus [33]) was a particular challenge to students. Initially, students of two classes modeled behavior for event-driven communication.

The students of all classes reported to have made more *conceptual* mistakes on modeling structure and behavior than on implementing behavior using GPLs ($Q3$). Interestingly, only in the intermediate questionnaires after four weeks of $C2$ and $C3$ students reported to have made more conceptual mistakes using GPLs.

We also asked students to estimate the efforts to understand MDE and GPL artifacts ($Q6$). In general, understanding the MDE artifacts of other team members was rated slightly more difficult than understanding GPL artifacts. Estimations changed in $C2$ and $C3$, where the middlewares became

TABLE II
THE FIRST PART OF OUR QUESTIONNAIRE INVESTIGATES THE EFFORT OF MODELING AND PROGRAMMING ACTIVITIES. QUESTIONS NOT PART OF THE INDICATED SURVEY ARE DENOTED BY "–".

Question/Answer	$C1_2$	$C2_1$	$C2_2$	$C3_1$	$C3_2$
1.) What percentage (0% - 100%) of the time did you spend on understanding ...					
C&C modeling	86.0	21.1	13.8	14.7	5.0
Behavior modeling	29.0	21.7	14.4	12.7	5.0
Code generation	33.3	11.8	11.8	16.2	6.4
LeJOS	10.7	15.0	–	–	–
Python	–	14.4	16.4	–	–
ROS	–	26.7	43.7	–	–
SmartSoft	–	–	–	40.0	46.8
2.) What percentage (0% - 100%) of the time did you spend on...					
modeling C&C structure	19.3	35.0	16.7	16.9	10.1
modeling behavior	36.4	14.4	18.9	6.1	4.8
implementing Java behavior	20.0	–	–	40.5	36.5
implementing Python behavior	–	45.0	64.4	–	–
building LEGO robots	24.3	–	–	–	–
implementing SmartSoft modules	–	–	–	30.2	23.3
3.) What percentage (0% - 100%) of the time was wasted because you tried something that was *conceptually wrong* (i.e., you tried something not supposed to be possible that way) while...					
modeling C&C structure	12.8	4.4	5.0	12.5	8.3
modeling behavior	33.6	3.3	9.4	8.5	7.9
building LEGO robots	40.7	–	–	–	–
implementing Java behavior	12.9	–	–	26.0	7.1
using SmartSoft modules	–	–	–	35.5	16.8
implementing Python behavior	–	33.9	8.3	–	–
using ROS modules	–	11.1	6.1	–	–
4.) What percentage (0% - 100%) of the time was wasted because you tried something that failed due to bugs in the code generator while...					
modeling C&C structure	25.7	8.9	14.4	12.5	8.1
modeling behavior	52.1	15.0	27.8	6.0	8.8
building LEGO robots	2.1	–	–	–	–
implementing Java behavior	5.7	–	–	24.6	9.4
implementing Python behavior	–	27.8	10.0	–	–
5.) How many times did you revise or recreate the...					
C&C models	1.3	1.9	3.0	3.0	11.0
Behavior models	1.8	0.7	2.9	0.5	1.2
LEGO robots	1.3	–	–	–	–
Java behavior implementations	0.4	–	–	7.6	12.2
SmartSoft modules	–	–	–	4.6	2.8
Python behavior implementations	–	1.6	6.1	–	–
ROS modules	–	0.2	3.0	–	–
6.) Rate from 1 (simple) – 10 (almost impossible) the effort to understand and work on artifacts created by your team members.					
C&C models	4.4	2.2	4.0	3.0	3.9
Behavior models	6.3	3.3	4.6	2.3	4.7
LEGO robots	2.9	–	–	–	–
Java behavior implementations	4.3	–	–	3.0	4.3
SmartSoft modules	–	–	–	7.3	8.8
Python behavior implementations	–	3.0	3.1	–	–
ROS modules	–	4.3	6.1	–	–

[9]Summer 2014 video: https://www.youtube.com/watch?v=TIspANC9TY4

TABLE III
THE SECOND PART OF OUR QUESTIONNAIRE INVESTIGATES THE STUDENTS' CONFIDENCE AND USAGE OF MODELS.

Question/Answer	$C1_2$	$C2_1$	$C2_2$	$C3_1$	$C3_2$
7.) Rate the amount of documentation 1 (no documentation) – 10 (well documented) of the artifacts of the team.					
C&C models	4.4	6.2	5.6	4.4	7.7
Behavior models	5.1	5.1	4.9	4.9	4.4
LEGO robots	2.6	–	–	–	–
Java behavior implementations	4.7	–	–	7.0	7.7
SmartSoft modules	–	–	–	3.3	2.1
Python behavior implementations	–	7.4	7.9	–	–
ROS modules	–	2.8	5.0	–	–
8.) Rate your confidence in the correctness of the artifacts created by you from 1 (no confidence) – 10 (works perfectly).					
C&C models	7.2	8.6	8.0	8.8	8.2
Behavior models	6.3	6.5	6.5	8.3	5.2
LEGO robots	7.4	–	–	–	–
Java behavior implementations	6.5	–	–	7.8	7.6
SmartSoft modules	–	–	–	4.6	7.4
Python behavior implementations	–	6.8	7.2	–	–
ROS modules	–	2.9	7.3	–	–
9.) Rate your confidence in the correctness of the... artifacts created by your team members from 1 (no confidence) – 10 (works perfectly).					
C&C models	7.1	7.8	7.9	9.0	7.2
Behavior models	6.1	6.1	6.1	7.8	6.3
LEGO robots	8.9	–	–	–	–
Java behavior implementations	6.6	–	–	7.1	7.2
SmartSoft modules	–	–	–	8.5	5.4
Python behavior implementations	–	6.7	7.7	–	–
ROS modules	–	4.5	6.0	–	–
10.) Rate the effort to fix bugs in the... artifacts 1 (simple) – 10 (almost impossible).					
C&C models	6.3	4.6	4.6	3.7	3.7
Behavior models	6.3	5.3	5.2	3.5	2.8
LEGO robots	4.7	–	–	–	–
Java behavior implementations	3.2	–	–	4.2	5.3
SmartSoft modules	–	–	–	8.5	8.2
Python behavior implementations	–	2.8	4.8	–	–
ROS modules	–	4.6	7.3	–	–
11.) How often did you run interactive tests with...					
C&C models	15.1	0.3	2.0	4.5	2.3
Behavior models	43.6	0.3	2.8	3.6	1.7
LEGO robots	33.3	–	–	–	–
Java behavior implementations	11.6	–	–	10.1	51.1
SmartSoft modules	–	–	–	1.0	7.2
Python behavior implementations	–	0.0	15.1	–	–
ROS modules	–	0.0	43.8	–	–
12.) How many non-interactive regression tests did you implement for...					
C&C models	0.0	2.0	5.6	0.0	14.2
Behavior models	0.0	0.3	1.1	0.0	0.0
LEGO robots	0.0	–	–	–	–
Java behavior implementations	0.3	–	–	4.3	20.6
SmartSoft modules	–	–	–	0.0	0.0
Python behavior implementations	–	1.5	6.2	–	–
ROS modules	–	0.0	0.3	–	–
13.) What percentage of the components you've developed does use automata?	57.3	–	10.2	–	6.0
14.) What percentage of the components do you think could have been developed using automata?	64.6	–	19.3	–	7.9
15.) How many days do you think would it take to reuse your solution with another (similar, e.g., Roomba) robot?	–	–	5.0	–	18.9
16.) Please order the technologies by the complexity to understand these:					
C&C models	–	–	2.9	–	3.6
Behavior models	–	–	3.8	–	3.3
Continuous integration	–	–	4.4	–	2.3
Python behavior implementations	–	–	1.9	–	–
ROS nodes	–	–	4.4	–	–
Maven	–	–	–	–	3.9
Java behavior implementations	–	–	–	–	1.7
SmartSoft modules	–	–	–	–	5.9

more powerful, and hence, more complex. In these classes, behavior modeling was considered as simple (or as complex) as implementing behavior with the respective GPLs and C&C modeling was considered even easier.

We observed an interesting difference between structural C&C modeling and behavior modeling. The students in all classes were most confident in the correctness of C&C models, followed by Java implementations and Python implementations ($Q8$ and $Q9$). This reception is independent of considering their own artifacts or the artifacts of colleagues. Interestingly, regression tests to provide confidence in correctness were mainly implemented for GPL artifacts only ($Q12$).

TABLE IV
THE THIRD PART OF OUR QUESTIONNAIRE WAS PERFORMED ONLY IN THE LATTER TWO PROJECT CLASSES IN THEIR RESPECTIVE SECOND SURVEYS AND INVESTIGATES DEVELOPMENT EFFORTS AND THE PROCESS.

Question/Answer	$C2_2$	$C3_2$
17.) Do you think Scrum helped to enable you develop the common code base? (Yes/No/Don't know)	7/0/1	9/0/3
18.) Rate the different parts of the established Scrum methodology 1 (totally useless) – 10 (great benefit). Rate 0 if you do not know what is meant by the proposed artifact / methodology.		
Product Backlog	5.9	5.3
Impediment Backlog	6.3	4.1
Sprint Backlog	5.1	5.7
Daily Scrum	8.6	7.9
Sprint Planning Meeting	9.1	8.5
Sprint Retrospective	5.6	6.2
Definition Of Done	6.0	5.5
19.) Rate the concrete implementation of the different parts of the Scrum methodology 1 (bad implementation) – 10 (best possible integration). Rate 0 if you do not understand the question.		
Product Backlog	6.5	3.5
Impediment Backlog	6.0	2.9
Sprint Backlog	4.9	4.1
Daily Scrum	8.2	7.3
Spring Planning Meeting	8.6	7.4
Sprint Retrospective	6.4	5.6
Definition Of Done	6.8	4.2

MDE tooling maturity: One challenge of MDE is the maturity of the available tooling. Students expressed that a considerable amount of their time was wasted due to bugs in the code generator ($Q4$). In addition, students reported that the lack of automated tests for MDE artifacts ($Q12$) results from lack of convenient automation (not models but the generated code had to be tested). The students of the first class $C1$ complained about (later improved) code generation times of up to five minutes.

MDE Motivation: From comparisons of GPL solutions vs. MDE solutions, some students were not convinced to apply MDE for behavior modeling. No student questioned C&C architecture modeling, but in classes $C1$ and $C2$ students asked why they should create automata when they could use GPLs to implement behavior. Nonetheless, the students were positive that their model-driven solutions could be reused with different platforms with little effort ($Q15$).

R2: Which development challenges result from CPS?

We have executed the case study three times with different robots and middlewares. These combinations posed different challenges to the students.

Middlewares: The class $C1$ employed Java and the educational LeJOS NXT middleware, which required less time to comprehend than ROS or SmartSoft ($Q1$). The students of $C2$ and $C3$ had to program more middleware-specific component implementations than the students of $C1$ ($Q2$). Moreover, these components were revised more often ($Q5$). Specifically, changes were necessary, when switching from simulators to real systems. In line with this, the effort to fix bugs in the middleware modules ($Q10$), which often amounted to guessing *magic numbers* (*i.e.,* for laser scanner configuration, navigation, or color sensor thresholds), was higher. This challenge was greater for ROS and SmartSoft than the LEGO NXTs. In interviews, the students of classes $C2$ and $C3$ also complained about the amount of technologies required to set up the development environment for the Robotino robots.

Hardware challenges: The students of $C1$ reported to have spent much time on designing and constructing the LEGO robots ($Q2$) to overcome their hardware limitations as well as their physical and mechanical challenges (*e.g.,* the lack of localization sensors or the challenge of creating enough force with the LEGO motors to enable pushing the coffee machine buttons with a mobile robot). In the beginning of $C2$, the students were provided with the new Robotino platform and a separated laser scanner. Connecting the laser to the platform again posed unforeseen hardware challenges to the students. Having attached the laser scanner, the students found that the uniform corridors of our department pose severe localization challenges.

Robot Availability: During $C2$, the students started using the robot later due to technical difficulties. They complained about the complex transition from a simulator to the real robot. In comparison, students of $C1$ appreciated that they were able to take their robots home and work on them during weekends.

R3: Which Scrum elements are perceived most beneficial?

In $C1$, where we first applied Scrum, we suggested daily Scrum meetings via Skype. The students denied this as too complicated to schedule with other activities. Later, their main complaints were lack of common development standards, heterogeneous level of information, and lack of communication.

The majority of students participating in $C2$ and $C3$ considered our variant of Scrum helpful to develop the common code base ($Q17$). In both classes, the 'daily scrum' (which actually was performed weekly) and the weekly sprint planning meetings were considered most helpful ($Q18$). In $C3$, 3 of 14 students were unsure about the benefits of Scrum, whereas in $C2$, only 1 of 9 was unsure about these (differences to participant numbers arise from students giving no answer to this question at all). This might be partly due to the team of $C3$ including students from bachelor programs without any previous software project experience (*cf.* Sec. IV-A).

To our surprise, the students of $C2$ considered the sprint backlog least useful, whereas the students of $C3$ considered the sprint backlog rather useful and the impediment backlog least useful. Both findings might be due to the respective backlog implementations as a wiki: the team of $C2$ rated the sprint backlog having the worst implementation and the team of $C3$ rated the impediment backlog having the worst implementation ($Q19$).

The weekly interviews showed that scaling Scrum to project class sizes might be challenging when an inexperienced Scrum master is also part of the development team. We tried to address this by defining two Scrum masters in $C2$ and three in $C3$. While the two Scrum masters of $C2$ cooperated in this role very well, the three Scrum masters of $C3$ reported to be very challenged by enacting their roles. Nonetheless, the students in all classes reported that they learned or improved working in agile software development teams.

R4: What should guide MDE and CPS technological choices?

Although LeJOS was considered less complex for programming the NXT robots, their construction required larger effort than expected ($Q2$). Nonetheless, the students of $C2$ and $C3$ spent much more time dealing with middleware challenges than the students of $C1$. Overall, for teaching purposes with a specific focus on modeling, the load with "accidental complexities" [17] imposed by middleware or platform challenges should be reduced. With the aim of teaching a realistic experience in MDE with CPS, we consider these complexities part of the learning.

Middleware maturity: The students of $C2$ and $C3$ complained about the available documentation and reported varying levels of frustration dealing with the middlewares' infrastructures. These elements were rated the most challenging to fix ($Q10$). Fortunately, the developers of SmartSoft at Ulm University of Applied Sciences were very helpful to assist the students of $C3$.

Hardware challenges: In line with the challenges identified from employing CPS, the students reported being unsatisfied with the sensor quality of the LEGO robots ($C1$) and were unhappy with the hardware challenges posed by connecting the laser scanner to the Robotino ($C2$).

Robot Availability: As mentioned in the previous section the availability of the CPS is an important factor. LEGO NXT robots ($C1$) can easily be obtained and replaced while larger robots ($C2$, $C3$) can only be used inside the university and defects risk success of the project.

A. Lessons Learned

Performing these case studies helped us to better understand the challenges of teaching agile MDE with CPS. For instance, fulfilling the teaching objective of using state-of-the-art development tools (such as the modeling languages, robot middlewares, and robot platforms) was as instructive as it was challenging (*e.g.,* due to lack of documentation or bugs). All students learned modeling software architectures and improved their agile teamwork skills. However, we might not have enforced testing enough as the amount of interactive tests and regression tests remained low throughout all three case studies. Overall, we consider the following lessons most crucial to performing similar project classes:

L1: Employ stable, well-documented platforms: Although realistic, development of CPS might come with complex hardware challenges (such as imprecise LEGO sensors or the complex Robotino laser scanner installation), we found the students' effort for this less instructive than expected. Research prototypes (MontiArcAutomaton and SmartSoft) and open source environments (ROS) might add a lot of accidental complexities for students that can quickly end in frustration.

L2: Prepare sufficient MDE teaching materials: Results from the questionnaires and interviews show that students found MDE challenging. In general, students had sufficient background in GPLs but much of MDE technology was new new to them. It is important to address this lack of knowledge either by adding a strong teaching component to the class or by requiring participation in MDE lectures.

L3: Adequately adjust Scrum for students: Adjusting Scrum to weekly meetings and employing teams of Scrum masters what parallely acted as developers was considered beneficial. However, the Daily scrum and weekly sprint planning meetings were considered significantly more useful by the students than the additional Scrum artifacts. We believe this is due to the additional management effort, which is justified by the aim of teaching agile development and not just *coding*.

L4: Align employed technology with teaching goals: The students of both $C2$ and $C3$ spent notable time with middleware challenges at the cost of modeling activities. When designing such a class, consider the employed MDE and CPS technologies carefully to reflect teaching goals. Finding the right level of complexity is however not easy and depends on many factors.

L5: Be aware of the limitations of educational CPS: The LEGO NXT hardware (*e.g.,* the quality and reliability of sensors) restricts the realism of robotics projects. The students of $C1$ put considerable effort in designing a navigation system based on black lanes and adhesive tape. Whether third-party sensors mitigate this, has to be evaluated.

VII. Threats to Validity

Our case studies have been executed without a control group. This raises threats to the case study's internal validity (causality) and to its external validity (generalizability). Results from questions we explored, *e.g.,* using MDE instead of GPLs, could help design such controlled experiments.

Threats to these case studies' internal validity arise from the participants lacking previous modeling experience. Instead the students had previous classes on Java, which interfered with estimating the benefits of MDE over GPLs. This selection bias probably manifested in the greater confidence in Java artifacts compared to behavior models. Also, the instruments to evaluate the students' experience are subject to issues: surveys induce biases and interviews were informal. Furthermore, participants might interpret the Likert scales differently and may have avoided extreme responses (central tendency bias). Moreover, the students provided answers based on their self-perception only. Following [5], we tried to mitigate this bias through interviews. However, as these project classes were on MDE for CPS and the students knew that they were graded, the case studies may also be subject to compensatory rivalry among the students. As the study did not separate the students into control group and experimental group, the effect of such rivalry can be hardly estimated.

Threats to external validity arise from (1) performing the project classes in the educational context of RWTH Aachen University where the students participated for a graded certificate; (2) the number of participating students; (3) the interviews being conducted by the supervisors instead of more neutral interviewers; and (4) applying the lessons learned in one project class to the subsequent classes (*e.g.,* number of Scrum masters, changes in platform and GPL), hence the classes become less comparable.

Threats to construct validity arise from employing surveys. The style of questions and answer options might have influenced the answers. For instance, using Likert scales from 1 to 10 might have been to fine grained for some answers. Also querying for percentages often produced answers with sums greater than 100%. Using other forms of data collection (such as commit inspection or online forms) could have produced more direct, detailed, or validated results.

Whether the observations of these three case studies are reproducible with other groups or whether the same students would perform differently in other projects is hardly predictable in a university context. Moreover, the project classes were performed in different semesters. Hence, the modeling techniques and tooling advanced from one class to another.

VIII. Conclusion

We have reported results from an exploratory study on teaching agile MDE for CPS with realistic challenges. The study spans three project classes in different semesters at RWTH Aachen University. In these classes, the students employed a variant of Scrum tailored to their roles as part-time developers. The teams successfully engineered complex robotics applications employing modern modeling techniques, robot platforms, and middlewares. To understand the effect of teaching agile MDE for CPS, we instrumented the project classes with interviews and surveys. Through these, we uncovered challenges of teaching agile MDE in the context of CPS. We documented these challenges to support future project class designs.

Appendix

Topics prepared by the students and presented in the kickoff meetings.

A. *NXT Java Coffee Delivery (Winter 2012/13)*

- MontiCore [31]: parser generation, language composition, code generation (1 student each)
- MontiArcAutomaton [21]: C&C modeling, behavior modeling with automata, code generation (1 student each)
- Tooling incl. SVN and SSElab platform (1 student)
- LeJOS LEGO Java middleware (1 student)
- Mavenbuild manager (presented by supervisor)
- Scrum [20] (presented by supervisor)

B. *Robotino ROS Python Transport Service (Winter 2013/14)*

- MontiCore [31]: overview, parser generation, language composition, code generation (1 student each)
- MontiArcAutomaton [21]: modeling overview, code generation (1 student each)
- Robotino platform and ROS [29] (2 students)
- Maven build manager (1 student)
- Scrum [20] (presented by supervisor)

C. Robotino SmartSoft Java Transport Service (Summer 2014*)*

- MontiCore [31]: overview (2 students), language composition (2 students), code generation (1 student)
- MontiArcAutomaton [21]: C&C modeling (1 student), behavior modeling with automata (2 students)
- SmartSoft middleware [30] (2 student)
- Robotino platform and simulator (1 student)
- Scrum [20] (1 student)
- Maven build manager (1 student)

One student joined the class after topic assignments and did not prepare a presentation.

References

[1] S. H. Kim and J. W. Jeon, "Introduction for Freshmen to Embedded Systems Using LEGO Mindstorms." *IEEE Transactions on Education*, vol. 52, no. 1, pp. 99–108, 2009.

[2] D. J. Barnes, "Teaching introductory java through lego mindstorms models," in *ACM SIGCSE Bulletin*, vol. 34, no. 1. ACM, 2002, pp. 147–151.

[3] F. Klassner, "A Case Study of LEGO Mindstorms' Suitability for Artificial Intelligence and Robotics Courses at the College Level," in *ACM SIGCSE Bulletin*, vol. 34, no. 1. ACM, 2002, pp. 8–12.

[4] L. Pareto, "Teaching Domain Specific Modeling," in *Proceedings of the 3rd Educators Symposium at the 10th International Conference MODELS 2007*, 2007, p. 7.

[5] P. J. Clarke, Y. Wu, A. A. Allen, and T. M. King, "Experiences of Teaching Model-Driven Engineering in a Software Design Course," in *Online Proceedings of the 5th Educators' Symposium of the MODELS Conference*, 2009, pp. 6–14.

[6] A. Hamou-Lhadj, A. Gherbi, and J. Nandigam, "The impact of the model-driven approach to software engineering on software engineering education." in *ITNG*, 2009, pp. 719–724.

[7] J. O. Ringert, B. Rumpe, and A. Wortmann, "A Case Study on Model-Based Development of Robotic Systems using MontiArc with Embedded Automata," in *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme*, H. Giese, M. Huhn, J. Philipps, and B. Schätz, Eds., 2013, pp. 30–43.

[8] J. Porubän, M. Bačíková, S. Chodarev, and M. Nosál', "Pragmatic Model-Driven Software Development from the Viewpoint of a Programmer: Teaching Experience," in *2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2014, pp. 1647–1656.

[9] B. Bruegge, S. Krusche, and L. Alperowitz, "Software engineering project courses with industrial clients," *ACM Transactions on Computing Education (TOCE)*, vol. 15, no. 4, p. 17, 2015.

[10] M. Missiroli, D. Russo, and P. Ciancarini, "Learning Agile Software Development in High School: an Investigation," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 293–302.

[11] K. Beck, M. Beedle, A. Bennekum *et al.*, "Agile Manifesto `http://agilemanifesto.org/`."

[12] B. Rumpe, *Modeling with UML: Language, Concepts, Methods*. Springer International, 2016.

[13] P. Cuenot, D. Chen, S. Gerard, H. Lönn, M.-O. Reiser, D. Servat, C.-J. Sjöstedt, R. T. Kolagari, M. Törngren, and M. Weber, "Managing Complexity of Automotive Electronics Using the EAST-ADL," in *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*. IEEE, 2007, pp. 353–358.

[14] J. Hutchinson, M. Rouncefield, and J. Whittle, "Model-Driven Engineering Practices in Industry," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, May 2011, pp. 633–642.

[15] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What Industry Needs from Architectural Languages: A Survey," *IEEE Transactions on Software Engineering*, 2013.

[16] J. Whittle, J. Hutchinson, and M. Rouncefield, "The State of Practice in Model-Driven Engineering," *Software, IEEE*, vol. 31, no. 3, pp. 79–85, 2014.

[17] R. France and B. Rumpe, "Model-Driven Development of Complex Software: A Research Roadmap," in *Future of Software Engineering 2007 at ICSE.*, 2007.

[18] E. A. Lee, "CPS Foundations," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 737–742.

[19] J. Bezivin, R. France, M. Gogolla, O. Haugen, G. Taentzer, and D. Varro, "Teaching modeling: why, when, what?" in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2009, pp. 55–62.

[20] K. S. Rubin, *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional, 2012.

[21] J. O. Ringert, A. Roth, B. Rumpe, and A. Wortmann, "Language and Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems," *Journal of Software Engineering for Robotics*, 2015.

[22] P. B. Lawhead, M. E. Duncan, C. G. Bland, M. Goldweber, M. Schep, D. J. Barnes, and R. G. Hollingsworth, "A Road Map for Teaching Introductory Programming using LEGO Mindstorms Robots," in *ACM SIGCSE Bulletin*, vol. 35, no. 2. ACM, 2002, pp. 191–201.

[23] M. W. Lew, T. B. Horton, and M. Sherriff, "Using LEGO MINDSTORMS NXT and LEJOS in an Advanced Software Engineering Course," in *CSEE&T*, 2010, pp. 121–128.

[24] P. Brosch, G. Kappel, M. Seidl, and M. Wimmer, "Teaching model engineering in the large," in *5th Educators' Symposium in conjunction with the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009), Denver, CO, USA*, 2009.

[25] J.-P. Steghöfer, E. Knauss, E. Alégroth, I. Hammouda, H. Burden, and M. Ericsson, "Teaching agile: addressing the conflict between project delivery and application of agile methods," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 303–312.

[26] N. Medvidovic and R. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, 2000.

[27] B. Rumpe, "Formale Methodik des Entwurfs verteilter objektorientierter Systeme," Doktorarbeit, Technische Universität München, 1996.

[28] J. O. Ringert, B. Rumpe, and A. Wortmann, "Transforming Platform-Independent to Platform-Specific Component and Connector Software Architecture Models," in *2nd International Workshop on Model-Driven Engineering for Component-Based Software Systems (ModComp) 2015*, ser. CEUR Workshop Proceedings, vol. 1463, Ottawa, Canada, September 2015, pp. 30 – 35.

[29] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[30] C. Schlegel, A. Steck, and A. Lotz, "Model-Driven Software Development in Robotics : Communication Patterns as Key for a Robotics Component Model," in *Introduction to Modern Robotics*. iConcept Press, 2011.

[31] H. Krahn, B. Rumpe, and S. Völkel, "MontiCore: Modular Development of Textual Domain Specific Languages," in *Proceedings of Tools Europe*, 2008.

[32] R. Heim, P. Mir Seyed Nazari, J. O. Ringert, B. Rumpe, and A. Wortmann, "Modeling Robot and World Interfaces for Reusable Tasks," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)*, 2015.

[33] M. Broy and K. Stølen, *Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement*. Springer Verlag Heidelberg, 2001.