[CMNR24] J. Charles, J. Michael, L. Netz, B. Rumpe:
Teaching Model-Driven Low-Code Development Platforms.
In: Educators Symposium, MODELS Companion '24:
International Conference on Model Driven Engineering Languages and Systems,
pp. 115-122, Association for Computing Machinery (ACM), Oct. 2024.

Teaching Model-Driven Low-Code Development Platforms

Joel Charles
charles@se-rwth.de
Software Engineering, RWTH Aachen University
Aachen, Germany

Judith Michael
michael@se-rwth.de
Software Engineering, RWTH Aachen University
Aachen, Germany
larst@affiliation.org

Lukas Netz
netz@se-rwth.de
Software Engineering, RWTH Aachen University
Aachen, Germany

Bernhard Rumpe
rumpe@se-rwth.de
Software Engineering, RWTH Aachen University
Aachen, Germany

Abstract

Low-code development platforms (LCDPs) are becoming increasingly important in industry, which confronts us in academic teaching with the challenge of educating students in the basic principles, critical engagement, and evaluation of LCDPs. This leads us to the question, how to teach the usage of different LCDPs during an university course. The short time frame of university-level courses makes it challenging to teach more than only one LCDP. In our teaching approach, students use two different LCDPs and create a web-application with both of them. Firstly, we require the students to define a target application with common modeling languages, next they use the first LCDP, at about half the time they switch to the second LCDP and present their findings of the differences in methodology and development processes at the end. We discuss this approach, show survey results from the participants, and explain lessons learned. This concept allows students critical engagement with LCDPs and model-driven software engineering. Supervisors get an insight into the learnability of each LCDP and how novices adapt to different domain-specific languages and their notations.

CCS Concepts

• **Software and its engineering** → **Model-driven software engineering**; • **Social and professional topics** → **Software engineering education**.

Keywords

Low-Code Development Platforms, Education, University-Level Courses, Model-Driven Software Engineering, Problem-Based Learning

ACM Reference Format:
Joel Charles, Judith Michael, Lukas Netz, and Bernhard Rumpe. 2024. Teaching Model-Driven Low-Code Development Platforms. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3652620.3687805

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687805

1 Introduction

Low-Code Development Platforms (LCDPs) are gaining significant importance in industry [31, 39, 41]. This presents us with a challenge in academic teaching on how to educate students in the basic principles, critical handling, and evaluation of LCDPs. The gap between the problem domain, which domain experts understand, and the software implementation predominantly understood by software engineers is still a challenging field of research [19]. LCDPs help to narrow this problem-implementation gap, as they empower domain experts to take an active role in the development process. In industry, a broad variety of LCDPs exist, e.g., A12 [30], Appian [5], BESSER [4], Mendix Platform [23], MontiGem [1], Nintex[35], OutSystems [25], or Pega Platform [37]. The popularity of this type of development platform results in the need to have software engineers who are able to develop, operate, and maintain LCDPs. This results in the need to educate students in the basic principles, critical engagement, and evaluation of LCDPs. Existing publications focus either on the comparison of different LCDPs regarding their functionalities or tool support [36, 40] or provide insights into the teaching of one specific LCDP [24, 26, 28]. To the best of our knowledge, there are no approaches published that allow students to compare platforms by using them to develop concrete applications.

The question we tackle is *how to teach the usage of different model-driven LCDPs during a university course.* Even though the short time frame of university-level courses (14 weeks in one semester) makes it challenging to teach more than only one LCDP, it was important for us that the students were able to work with at least two platforms to be able to compare them and thus derive a broader understanding of LCDPs instead of only learning the usage of one specific platform. Students were at least in their 4^{th} semester, which means they already had several courses where they had learned traditional software development with Java. As their development skills are not well established at this point in time and with the given time limit, the focus of the study could not be the comparison between LCDP and a traditional approach but had to be between different LCDPs.

In this paper, we propose a course structure and an established procedure that allows students to experience the concepts of LCDPs in an Model-Driven Software Engineering (MDSE) project, share

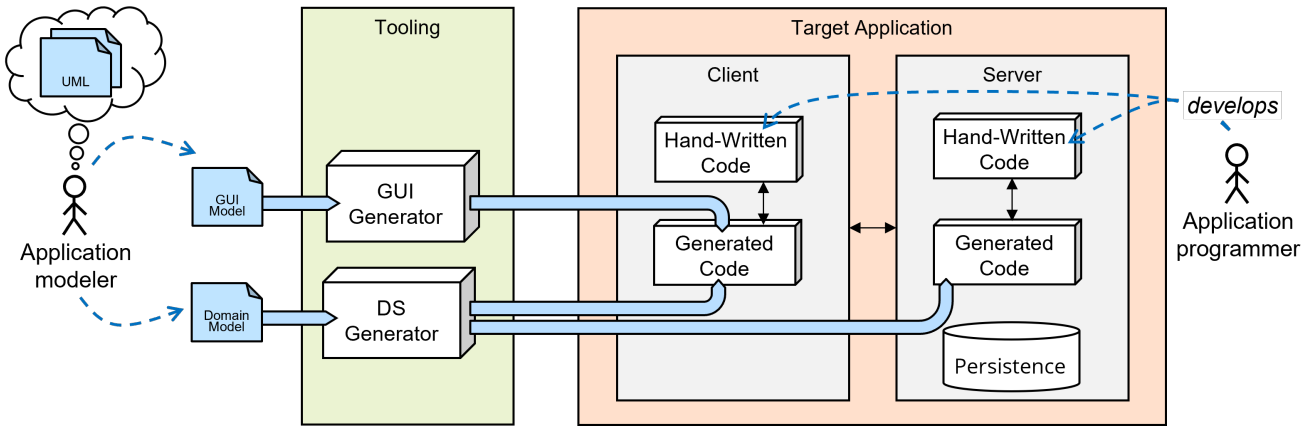

Figure 1: Basic MontiGem Architecture, Roles and Development Process

accumulated experiences with each other, and immediately experience an alternative approach for software development. Students gained a critical understanding of LCDPs by practically pushing the limits of each platform and learning about the individual differences of LCDPs and their Domain-Specific Languages (DSLs). To help others make an informed decision about how to conduct their teaching with LCDPs, we discuss our and the students' experiences with this teaching concept and present students' feedback.

The paper is structured as follows: The next section explains the research context and related work. Section 3 presents the project class structure in different phases and its execution. Section 4 addresses the survey accompanying the course. Section 5 discusses lessons learned before the last section concludes the paper.

2 Preliminaries & Related Work

We introduce the concept of LCDPs and their relationship to model-driven software engineering and introduce the utilized LCDPs MontiGem and A12.

Low-Code Development Platforms. According to [10], low-code development is a success story for MDSE. Both, low-code software development and MDSE, are strongly related to each other, as they try to reduce the problem-implementation gap [19] and rely on abstraction methods. Di Ruscio et al. [14] examine the differences and commonalities of low-code software development and MDSE in detail. For the purpose of this paper, we are particularly interested in the low-code and model-driven approaches that overlap in their characteristics. Di Ruscio et. al [14] state that a characteristic of low-code software development is the aim to reduce the amount of hand-coding required to realize a system. This characteristic is shared with some model-driven approaches [15, 16]. However, the reduction of hand-written code can arise from the use of models, data, or schema-less XML/JSON documents. To leverage low-code software development to a low-code development platform, support for deployment and life-cycle management capabilities is needed. Bock and Frank [3] highlight the feature variance between seven LCDPs represented on the market, e.g., data structure and workflow specifications, connection to external APIs, or graphical editors. A LCDP often features a graphical user interface for defining the target software, e.g., through models [13, 44]. LCDPs may produce entirely operational application software, but it may require developers to add a 'low' amount of hand-written code. The abstraction of low-level concerns empowers citizen developers to contribute their expertise in a machine-processable manner [3, 7, 10].

Only a few approaches describe how to use LCDPs or low-code approaches in education mainly using only one LCDP in teaching. Wang and Wang [43] present a teaching methodology and pedagogical concept for one *no-code* development platform. Similar to our approach a preliminary assessment was performed after the course, but the focus lies within the target application and not in the education on *low-code* development. Poe and Mew [38] use the LCDP Mendix to teach agile methods of software development. In contrast to our approach, is the course focused on teaching the agile method, such as scrum and cooperative software development, rather than the LCDP or MDSE. Tisi et al. [42] present a concept to teach 15 PHD students cloud-based LCDPs. The authors focus in this publication on the engineering challenges of a single platform rather than educational concepts for classes. Adrian, Hinrichsen, and Nikolenko [2] present a concept for a teaching unit within an industrial engineering program focusing on app development. Metrôlho et al. [27] give insights into an ITC training designed to reskill unemployed people who mainly have already attained a higher education level in the STEM area. In these trainings, they use only the LCDP OutSystems. In another work, Metrôlho et al. [26] use the LCDP OutSystems in a software engineering course for students in Scrum processes. Again, only one concrete LCDP was used. Lebens and Finnegan [24] present a course on agile development where the LCDP Microsoft Power Apps platform was used to create a web application. Their aim with using a low or no-code development platform was to enable students to spend less time learning to create apps and rather focus on getting an understanding of how agile processes work. Mew and Field [28] use the LCDP Mendix to teach an undergraduate project management course in the information systems program. Fernandes et al. [17] report on their findings on using the LCDP OutSystems for their project-based learning approach in a software engineering course.

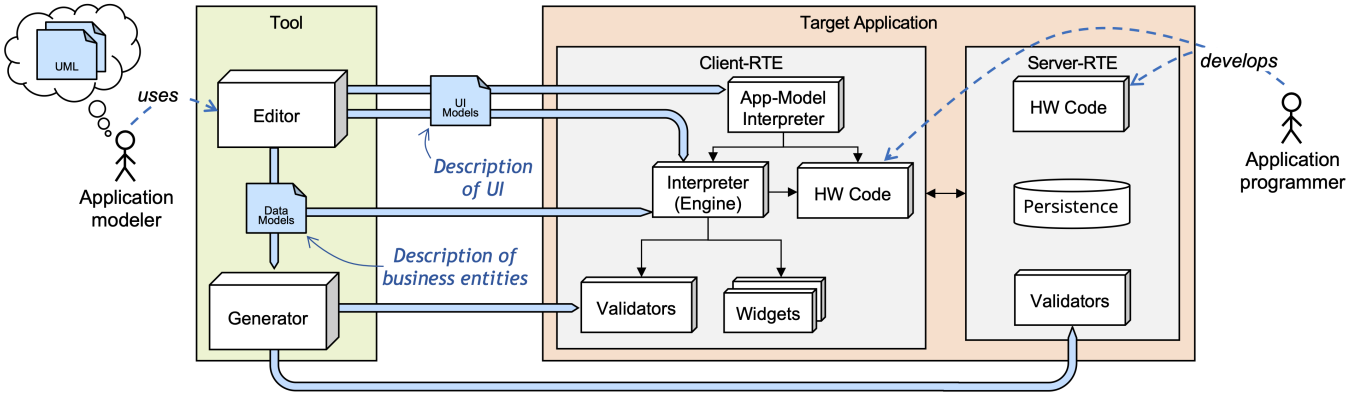

Figure 2: Basic A12 Architecture, Roles and Development Process

Within our university-level course, we use the LCDP MontiGem and the LCDP A12 to teach two different low-code platforms and enable the students to compare different LCDPs. The following provides an overview of their usage from the modeling perspective.

LCDP MontiGem. MontiGem is a generator framework [1, 22] that creates a data-centric web-based information system [8], e.g., for financial controlling [20], process-aware information systems [16], assistive systems [32, 33], IoT app stores [9], digital twin cockpits [12], or LCDPs for digital twins [11]. The target application is a three-tier server-client application, with an Angular (Typescript) based front end and a Java-based TomeEE back end. The model-driven architecture of the target application is defined by the input of the generation process: a class diagram for domain concepts and Graphical User Interface (GUI) models.

An *application modeler* defines a set of GUI and domain models, setting up the data structure and user interfaces of the target application (see Figure 1). MontiGem consists of two major generators. The first one is a data structure generator, which processes class diagrams and generates a database and basic infrastructure for both the server and client to access and move data from and to the database and between each other. The second one is a GUI generator, which processes the GUI models [21, 22], that can reference data structures defined in the input class diagrams. Each GUI model defines a new web page in the example application.

If needed, the application can be customized by hand-written code, that is developed by the *application programmer*. It extends the generated code, both in server and client, adapting or replacing the generated functions.

LCDP A12. The enterprise LCDP A12 [30] is developed by the company mgm technology partners and used in customer projects for the application domains insurance, e-commerce, and the public sector. Applications realized with A12 are typically single-page web applications, based on a client-server architecture. Software development within A12 is divided into two parallelizable branches.

The *application modeler* begins to set up the modeling tools via a provided installer (see Figure 2). He uses visual model editors to describe domain specifics in data and UI models. Domain-specific constraints (validation logic) can be formulated using a DSL. Validation code is generated, that can validate data consistency at runtime. Due to A12's focus on enterprise business applications mainly forms and documents are processed. Within UI models, data models are referenced and mapped to UI components. Within the target application, the models can be interpreted at runtime by *engines*. They orchestrate reusable low-level UI components (widgets) such as a data picker to render the modeled UI [29]. The engines utilize the generated validation code along with the UI models to automatically visualize incorrect entries. An *application model* defines the layout structure of the UI and its behavior, and which UI component is displayed. E.g., a tabular overview of the data is displayed to the left of the details for a record in a form. The models are deployed to the target application using the modeling tool.

In parallel, the *application programmer* is provided with a project template. The template is a functional client-server application with access management which establishes a recommended project structure. Sample models are used to demonstrate the integration of A12 models. It allows for the integration of third-party libraries and other aspects via handwritten code. The *application programmer* registers the UI components referenced in the application model in the template. They can either be taken from the A12 widgets library or developed independently. In addition, handwritten code allows for an optional customization of the engine's interpretation of the models in case there are special requirements. The result is an executable, tailored A12 application.

3 Teaching Method & Execution in a Software Engineering Lab

Our teaching method is based on a software engineering lab and relies on problem-based learning [6, 18] to teach MDSE with LCDPs.

The variation between LCDPs requires teaching to differentiate between platform specifics and generic low-code approaches. In a university context, there are specific external constraints within which the methodology may be chosen. Our objective is to teach a mixed group of bachelor and master students of computer science (or closely related study programs) the usage of model-driven LCDPs. The course aim is to be fulfilled within the framework of a semester lab (14 weeks, see lab structure in Figure 3). Successful completion of a programming lecture is a prerequisite for participation in our course. Furthermore, all students had initial modeling

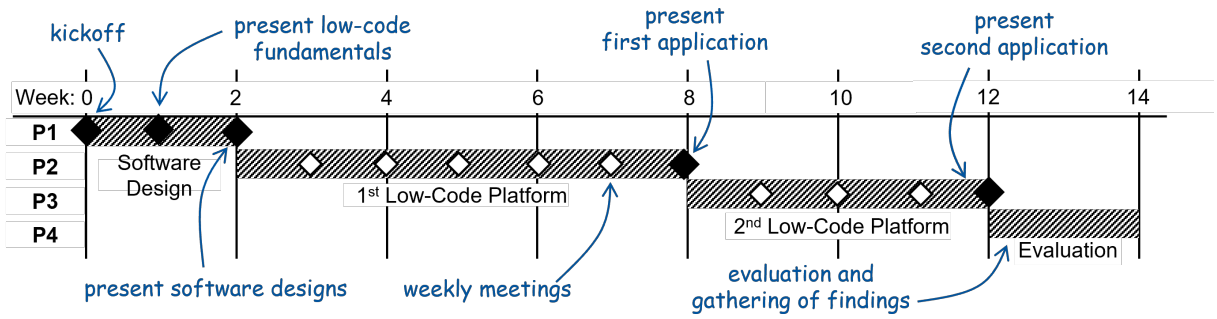

Figure 3: Project classes can be divided into four phases (P1-P4). Black diamonds indicate, a meeting where students give a presentation, and white diamond represents a regular meeting, where a student can get support.

experience with UML in previous semesters. However, this does not imply that the students are experienced full-stack developers. We have developed a methodology to convey the teaching objectives.

By teaching two LCDPs in the lab, we aim to prevent platform specifics from being misconstrued as generic LCDP functionality by the students. We have chosen the LCDPs A12 and MontiGem as representatives for two model-driven LCDPs. They were selected firstly because they differed sufficiently in the modeling phase and runtime (e.g. A12: generation & model interpretation at runtime; MontiGem: pure generative approach) as well as the technology stack. Secondly, the supervisors had prior expertise in the platforms. Due to the external constraints of the course, a time frame must be adhered to. By dividing the course into four phases with their respective teaching objectives and deadlines, adherence to the time frame and objectives is encouraged.

Once students have progressed through the phases, they should have modeled and developed a web application using two different LCDPs. In particular, students are expected to focus not only on the development as such but also on the development process within the LCDPs. Conclusions should be inferred about the differences between the platforms and low-code development in general. By engineering the same application with two LCDPs we expect to be more time-efficient, as we only need one modeling phase instead of two. In addition, it is easier for the students to compare both platforms and draw generalized conclusions about LCDPs.

3.1 Phase 1: Requirement Analysis and Software Design

The intended deliverables of this phase are a set of models that describe the target application to be implemented. To achieve the objectives, the heterogeneous levels of knowledge of the participants concerning the required technologies (e.g. web development) are aligned. The students are asked to assess their knowledge of software development and familiarity with LCDPs, and indicate a preferred grouping. At the beginning of the first phase the students have to recapitulate and present fundamentals of the frameworks/libraries used in the LCDPs, that will be needed in the upcoming phases. The presentation of the topics in the first week is intended to equalize the knowledge levels of the participants and to stimulate collaboration. This process helps the supervisors to establish a balanced set of groups in which the tasks are processed. In general, due to the completed courses in their computer science curricula, the students are expected to have a basic understanding of Model-Driven Software Engineering and basic programming skills.

In the run of the lab, 16 participants were registered. Half indicated that they would like to be in a group with a preferred partner. The self-assessed skill sets ranged from beginner-level programmers to individuals with a lot of experience and who had already used LCDPs. The students could be divided into a set of four groups with comparable skill sets.

In a subsequent step, the students are introduced to the overarching theme of the target software. In our case, this was an online store offering goods that the students could define. Students were free to choose the domain of their store and came up with the following ideas:

App Theme: Block Chain Store
Motivation: An online shop sells two item categories: blockchains and chain blocks.
Approach: Students defined a data structure for shop clerks and clients and added different types of blockchain and chain blocks and the shopping process (cart, payment, shipping).

App Theme: Boston Consulting Fruit
Motivation: A web application in which multiple consulting services can be purchased.
Approach: The students modeled a data structure to set up a consulting team with their respective skills and defined classes to handle fictive consulting forms.

App Theme: Online Corner Shop
Motivation: A shop that sells all kinds of everyday items.
Approach: The students modeled a data structure for shop clerks and clients. Shop clerks could edit items, e.g. change stock or prices.

App Theme: Tea Shop
Motivation: A web shop specializing in Tea.
Approach: The students modeled a data structure defining a tea warehouse. Tea was not modeled as separate items, but purchased by weight. Similar to the other apps, there were two types of users, management, and clients.

Based on their store, the students have to provide requirements and design models that define the store and its features. Before their modeling attempt, they are presented with a use case to provide a reference for the size and complexity of the models. The students are not bound to specific modeling languages, however, they are required to define the data structure, features, and access permissions for users interacting with the application. Thus, students are encouraged to use modeling languages and modeling paradigms learned in previous lectures. To prevent participants from being influenced by the specific functional scopes of the LCDPs, they are not yet introduced to the LCDPs used.

At the end of the first phase, the students present and discuss the developed models and requirements for each shop. All groups chose class diagrams to describe the data structure and described intended features with bullet point lists. Although Feature Diagrams, Use Case Diagrams or BPMN provide the means to define large aspects of an application, the four groups used more informal ways of representing functionalities, features, and processes. The meetings in the first phase (Figure 3) are used to refine and optimize the models and adjust the planned software's complexity and difficulty. At this point, the supervisors must have experience with the LCDPs that should be used by the students later, as they need to be able to evaluate whether a model is feasible to be implemented with the provided platforms in the given time.

3.2 Phase 2: First LCDP

In phase 2, the groups are assigned one of two LCDPs in which they are supposed to replicate their modeled web-application as closely as possible. It is ensured that each of the platforms is used by half of the groups. At the end of the second phase, the students present their progress. Additionally, the students present the challenges they encountered during the development process and discuss how they managed to overcome those. The hints can be used by the groups in the subsequent phase. In case all requirements defined by the models are realized within the given time frame of phase 2, the students can add further functionality and apply software engineering best practices, such as automated testing.

During the lab execution, the meetings have been separated according to the LCDP used in the phase. For this lab, we cooperated with an industry partner, mgm technology partners, to provide the students with two LCDPs with different modeling notations. Each group gets an introduction to their platform by their supervisors. Subsequently, students are required to work with the platform and realize the system they have previously defined. For support, weekly question and answer sessions are offered in addition to the possibility of contacting the supervisors at short notice.The meetings addressed arising challenges and progress was discussed.

Within the time frame, no group was able to completely realize their planned application. However, many achieved a system having basic functionality such as user management, shop item management, and a rudimentary shopping process. This was combined with user interfaces in varying levels of sophistication. In their presentations, they highlighted the features of their application as well as the gaps in their requirements and models, and the process that was used to create the software. At this point, the first differences between the two LCDPs were discussed. For example, this concerns the DSLs used by the platform and the platform's tooling. Other aspects include the code creation process (generative/interpretive), the amount and complexity of the required hand-written code, as well as the overall usability/learnability of the platform.

3.3 Phase 3: Second LCDP

In this phase, the students use the same requirements and models that were defined in the first phase but use a different LCDP to realize the web application. Similar to phase two, the students developed a web application as close to their plans from the first phase as possible. At the end of this phase, the final applications are presented and discussed. At this point, each student has attempted to develop a web application based on the same requirements and models in two different LCDPs. Thus, they can draw conclusions about the differences in the development processes and about low-code development in general. Among other topics, the subsequent discussions were held on the following topics:

- Pragmatism of the low-code approach for the specific presented use case
- Benefits and drawbacks of the development approaches
- Differences in capabilities of the platforms
- Differences in platform usability/learnability
- Differences in amount and complexity of required hand-written code

Based on the know-how gained from the second phase, and the feedback of their colleagues, we expected this development segment to progress faster and still reach comparable development results. The advantages of each group compared to the previous phase were:

- The gap between the proposed model (Phase 1) and the actual data structure of the target application is now known. Thus, it was expected that the students need fewer iterations to develop the data structure for the second low-code application and they would spend less time on creating models.
- The students have heard experiences from their colleagues who have already used the platform. They know in advance which challenges they will face and how to overcome them.
- The students have established a general development concept for the application, resulting in less time to spend on figuring out what to do and how to do it.

In the kick-off meeting of this phase, each group gets an introduction to their respective platform. In this meeting, the questions of the students from the last introduction, and weekly meetings from the previous phase were included to streamline the process of getting familiar with the platform. Similar to the previous phase, there were weekly meetings, in which questions of the students to each platform were answered and progress was ensured. The students reached comparable development results despite a shorter amount of time for the second platform.

At the end of the third phase, the students present their second version of their application and highlight the features and capabilities of their hand-in. At this point, each group has the experience of using two low-code platforms to develop two versions of one application idea. Figure 4 and Figure 5 shows an example, of the tea shop of Group D implemented in each platform. In this

phase, the groups were required to highlight the differences in the development process and approaches taken in both platforms.

3.4 Phase 4: Evaluation

The final phase contains a formal recording of the student's experiences with the different LCDPs. Therefore, we set up a study.

Study Design. The main goal of our study was to evaluate the teaching method. Additionally, we wanted to explore to what extent both LCDPs are suitable for this lab. The questions of the study focused on the different modeling techniques, code creation capabilities, and limitations of both platforms. The questionnaire included seven topics: (1) self-assessment (2) differences in low-code approaches (3) visual and textual modeling notations (4) different modeling languages (5) gaps between models and implementation (6) development process and methodology (7) provided tooling and debugging. The students were also allowed to give anonymous feedback on the structure and organization of the course. Finally, we used this phase to gather feedback from the students regarding the organization of the project class to provide a better learning experience in the next iteration of this project class.

4 Survey

At the end of the course, the students participated in two surveys: the general student course evaluation of the university and a specific survey targeting the exact teaching content. Both surveys were conducted anonymously in compliance with GDPR and were online available to the students. Seven students participated in the general student course evaluation and thirteen took part in the specific survey. Table 1 shows the asked questions and main results.

Survey Participants. The course is only part of computer science-related study programs. Therefore we expected that the participants have knowledge in programming and computer science. We included a self-assessment section in the survey as expert knowledge in computer science is often also gained outside the study program, e.g., employment as a software developer or hobby. The questionnaire was completed by 13 of the 16 students that took part in the lab. The study data can be found at [34]. Three participants are studying for a Computer Science Bachelor, the others are enrolled in a Master's program, either in Computer Science or a subject strongly related to it. About half claim to have five or more years of software development experience, whereas the remaining claim to have almost no experience (1 year or less). Three of the experienced software developers are experts in web development as well, whereas the others have little to no experience (1 year or less) in that particular field of software development. None of the students claim to be an expert in MDSE or low-code development.

Low-Code Platforms. The students were asked to answer questions about the LCDPs on a scale between 1 and 5: '1' in case they disagreed with the statement and '5' in case they agreed. The majority of the students perceived the LCDPs as helpful and could see time savings even for the relatively small target applications they created. Due to the nature of the modeling approach used, the students faced limitations in their means to define the application and had to add hand-written code or custom components. On average the students agreed that these limitations in modeling visual

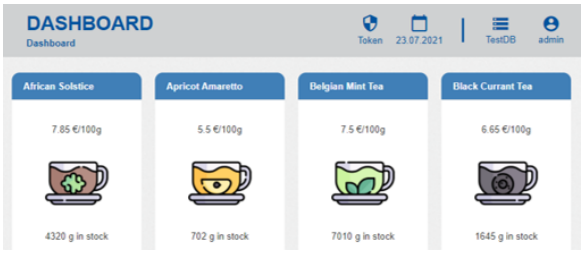

Figure 4: User Interface for a Tea shop, generated with MontiGem by Group D

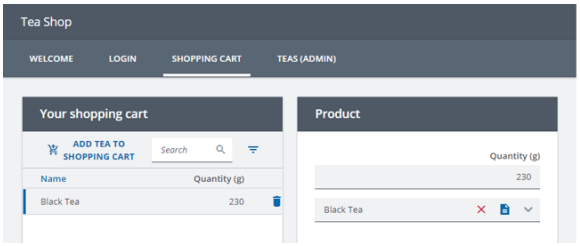

Figure 5: User Interface for a Tea shop, generated with A12

components, and data structures helped to ease the development process compared to less constrained implementation by hand.

Visual and Textual Modeling Methodology. The two platforms differ in the way they can be used to define models. One platform provides a lot of visual tool-support to design and configure models whereas the other platform relies on textual input and command line interfaces. In this context we refer to the methodology using visual aids and visual representations to review and define models as 'visual' and the methodology that requires the user to define and review the models on a text-only based representation as 'textual'. Table 1 shows questions from the survey, regarding the students' experiences with both modeling methodologies. The students could state which modeling methodology they perceived fitted best to answer the question, ranging from 1 "visual methodology" to 5 "textual methodology". The parameter in the last column of Table 1 is the arithmetic mean of the given answers.

There does not seem to be a clear tendency towards one specific modeling methodology (Table 1, Q1.1). The lack of visual aids results in easier copying and editing of given textual examples and tutorials (Q1.5, Q1.9), on the other hand, models with a visual notation help to create completely new models without the need for examples (Q1.4, Q1.7, Q1.8). The students grasped the information contained in a model faster when using a visual methodology (Q1.3, Q1.10). However, the purely textual variant benefits from easy integration into cooperative software development infrastructure and thus is perceived as easier to design cooperatively (Q1.11, Q1.12, Q1.13).

Student course evaluation. The students participated in an evaluation conducted by the university. They were asked to answer the questions Q2.1 - Q2.8 on a scale between 1 and 5: '1' in case they disagreed with the statement and '5' in case they agreed. The presented value in Table 2 is the median value over all answers provided by the university. Based on these answers, one can conclude

Table 1: Participant Questionnaire: Q1.x: Modeling Methodology, Q2.x: Student course evaluation. Answers from 1-5 with 1 visual and 5 textual. The results are the arithmetic mean.

#	Question	Results
Q1.1	I would rather use this modeling variant in a low-code platform	2,5
Q1.2	It is easier to analyze and understand small models in this modeling variant	3,5
Q1.3	It is easier to analyze and understand complex and large models in this modeling variant	2
Q1.4	It is easier to create completely new models in this modeling variant	2
Q1.5	It is easier to create new models based on existing ones in this modeling variant	4
Q1.6	It is faster to develop models in this modeling variant	3
Q1.7	It is less error-prone to develop models in this modeling variant	2
Q1.8	It is easier to understand what I can model with this modeling variant (e.g what elements can be used to create a model)	3
Q1.9	It is easier to apply learned concepts from documentation in this modeling variant	4
Q1.10	It is easier to navigate in complex and large models in this modeling variant	3
Q1.11	It is easier to comprehend changes in models in this modeling variant	4
Q1.12	It is easier to develop models cooperatively in distributed teams in this modeling variant	4
Q1.13	In general I prefer tooling based on this modeling method	4

Table 2: Student course evaluation. Answers from 1-5 with 1 disagreeing and 5 agreeing. Results are the arithmetic mean.

#	Question	Results
Q2.1	The course is interesting	4
Q2.2	During the lab I can apply what I have learned.	4
Q2.3	The trials improve my experimental skills.	4
Q2.4	Supervisors could assist with problems that arose	4
Q2.5	Materials provided were helpful	3
Q2.6	The preliminary talks regarding the experiments help me conduct the experiment.	4
Q2.7	The lab begins and ends on time.	5
Q2.8	The time invested in the work with the digital teaching material was appropriate for the learning objective.	4

that the students were mainly satisfied with the organization of the course but the materials provided should be more helpful.

5 Lessons learned

Despite differing levels of experience, we can conclude that the time-saving properties of the LCDPs enabled the students to create the applications while remaining within the time constraints of the course (Q 2.7, Q 2.8). Although the time frame with the second platform has been shorter, comparable results are achieved. We have to acknowledge that LCDPs require a certain level of experience (modeling/coding) with the platform. Each platform has its own concepts that need to be internalized. They differ in their sophistication and characteristics and depend on the positioning of the platform in the market. In general, low-code approaches aim to automate as much of the development process as possible to reduce the time and resources of the (citizen) developer. Students reported an easier and less error-prone implementation process by using LCDPs. They also reported that at least one of the used LCDPs shortened development time. In general, the students stated that they were not able to realize all planned models within the course. Every group created a functional web application however in order to have more flexibility both in modeling and in implementation stretch goals' were discussed with each group. Thus we have to state that the applications handed in by the students were not completed.

The students using the platform successfully to create concepts and models, discussing the platforms, and creating an application with them lead to the conclusion that the students learned how to develop software with model-driven LCDPs. As stated above, the proposed course is intended to teach students software development and familiarize them with LCDPs. In addition to the knowledge acquired through the practical implementation of the course, students reported learning success by being able to apply their knowledge and having improved on their skills (Q2.1- Q2.4).

The students would have preferred if the LCDPs would have been provided with graphical tooling (Table 1, Q1.1), although they have a general preference for text-based tooling (Q1.13), thus, they do not favor one specific variant to develop models as seen in Q1.6. Within the Computer Science curriculum, students are used to define models graphically and develop software with textual code editors. In the course, the students had to rely on both methodologies to create the web applications. We expected the students to prefer graphical editing tools. However, there was not a strong tendency towards any of the variants. The 'textual' variant was easier to edit cooperatively (Q1.11, Q1.12) and the examples from documentation were easier to adapt for the own use case (Q1.5, Q1.9). With visual aids and graphical editors, students reported getting a better overview of complex models (Q1.1 - Q1.4, Q1.7).

6 Conclusion

The proposed method teaches important findings about software engineering processes while providing an insight into the usage and capabilities LCDPs. When applying this teaching method at other universities, we suggest the following aspects based on our findings: (1) Use two LCDPs and choose one with a visual and one with a textual model representation to show students the differences in grasping information, copying and editing given examples, creating new models, and cooperative development. (2) Choose familiar platforms or ones that are easy to get started with as time is limited within the lab. Most of the time should be spent by the students on using the platforms rather than getting to know them. (3) LCDPs are well suited to teach software development not only in the context of smaller application prototypes but also for complete software such as web applications. (4) Supervisors must be familiar with the platforms and available to students on short notice. LCDPs

mantra of reducing hand-written code eliminates many programming errors. However, platform-specific modeling errors can occur while modeling, for which there are fewer proposed resolutions online than for general-purpose programming languages. For this reason, proper documentation and support is critical.

References

[1] Kai Adam, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. 2020. Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In *40 Years EMISA (EMISA'19) (LNI, Vol. P-304)*. GI, 59–66.

[2] Benjamin Adrian, Sven Hinrichsen, and Alexander Nikolenko. 2020. App Development via Low-Code Programming as Part of Modern Industrial Engineering Education. In *Advances in Human Factors and Systems Interaction*. Springer.

[3] Bock Alexander and Frank Ulrich. 2021. In Search of the Essence of Low-Code An Exploratory Study of Seven Development Platforms. In *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. ACM/IEEE.

[4] Iván Alfonso, Aaron Conrardy, Armen Sulejmani, Atefeh Nirumand, Fitash Ul Haq, Marcos Gomez-Vazquez, Jean-Sébastien Sottet, and Jordi Cabot. 2024. Building BESSER: An Open-Source Low-Code Platform. In *Enterprise, Business-Process and Information Systems Modeling*. Springer Nature Switzerland, 203–212.

[5] Appian. 2021. *Low-Code Application Development.* https://appian.com/platform/low-code-development/low-code-application-development.html

[6] Brigid JS Barron, Daniel L Schwartz, Nancy J Vye, Allison Moore, Anthony Petrosino, Linda Zech, and John D Bransford. 1998. Doing with understanding: Lessons from research on problem-and project-based learning. *Journal of the learning sciences* 7, 3-4 (1998), 271–311.

[7] Mariana Bexiga, Stoyan Garbatov, and João Costa Seco. 2020. Closing the Gap between Designers and Developers in a Low Code Ecosystem. In *23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Comp.* ACM.

[8] Constantin Buschhaus, Arkadii Gerasimov, Jörg Christian Kirchhof, Judith Michael, Lukas Netz, Bernhard Rumpe, and Sebastian Stüber. 2024. Lessons Learned from Applying Model-Driven Engineering in 5 Domains: The Success Story of the MontiGem Generator Framework. *Science of Computer Programming* 232 (2024), 103033. https://doi.org/10.1016/j.scico.2023.103033

[9] Arvid Butting, Jörg Christian Kirchhof, Anno Kleiss, Judith Michael, Radoslav Orlov, and Bernhard Rumpe. 2022. Model-Driven IoT App Stores: Deploying Customizable Software Products to Heterogeneous Devices. In *21th ACM SIGPLAN Int. Conf. on Generative Programming: Concepts and Experiences (GPCE 22)*. ACM.

[10] Jordi Cabot. 2020. Positioning of the Low-Code Movement within the Field of Model-Driven Engineering. In *23rd Int. Conf. on Model Driven Engineering Languages and Systems: Companion (MODELS '20)*. ACM. https://doi.org/10.1145/3417990.3420210

[11] Manuela Dalibor, Malte Heithoff, Judith Michael, Lukas Netz, Jérôme Pfeiffer, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. 2022. Generating Customized Low-Code Development Platforms for Digital Twins. *Journal of Computer Languages (COLA)* 70 (2022).

[12] Manuela Dalibor, Judith Michael, Bernhard Rumpe, Simon Varga, and Andreas Wortmann. 2020. Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits. In *Conceptual Modeling*. Springer, 377–387.

[13] G. Daniel, J. Cabot, L. Deruelle, and M. Derras. 2020. Xatkit: A Multimodal Low-Code Chatbot Development Framework. *IEEE Access* 8 (2020), 15332–15346. https://doi.org/10.1109/ACCESS.2020.2966919

[14] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. 2022. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling* 21, 2 (2022), 437–446.

[15] Imke Drave, Akradii Gerasimov, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. 2021. A Methodology for Retrofitting Generative Aspects in Existing Applications. *Journal of Object Technology (JOT)* 20 (2021), 1–24. https://doi.org/10.5381/jot.2021.20.2.a7

[16] Imke Drave, Judith Michael, Erik Müller, Bernhard Rumpe, and Simon Varga. 2022. Model-Driven Engineering of Process-Aware Information Systems. *Springer Nature Computer Science Journal* 3 (2022).

[17] João Paulo Fernandes, Ricardo Araújo, and Mário Zenha-Rela. 2020. Achieving Scalability in Project Based Learning through a Low-Code platform. In *XXXIV Brazilian Symposium on Software Engineering (SBES '20)*. ACM, 710–719. https://doi.org/10.1145/3422392.3422482

[18] Víctor M Flores Fonseca and Jesica Gomez. 2017. Applying active methodologies for teaching software engineering in computer engineering. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje* 12, 4 (2017), 182–190.

[19] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering (FOSE '07)* (2007), 37–54.

[20] Arkadii Gerasimov, Patricia Heuser, Holger Ketteniß, Peter Letmathe, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. 2020. Generated Enterprise Information Systems: MDSE for Maintainable Co-Development of Frontend and Backend. In *Comp. Proc. of Modellierung 2020 Short, Workshop and Tools & Demo Papers*. CEUR-WS, 22–30.

[21] Arkadii Gerasimov, Judith Michael, Lukas Netz, and Bernhard Rumpe. 2021. Agile Generator-Based GUI Modeling for Information Systems. In *Modelling to Program (M2P)*. Springer, 113–126.

[22] Arkadii Gerasimov, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. 2020. Continuous Transition from Model-Driven Prototype to Full-Size Real-World Enterprise Information Systems. In *25th Americas Conf. on Information Systems (AMCIS 2020) (AISeL)*. AIS, 1–10.

[23] Martin Henkel and Janis Stirna. 2010. Pondering on the key functionality of model driven development tools: The case of mendix. In *International Conference on Business Informatics Research*. Springer, 146–160.

[24] Mary Lebens and Roger Finnegan. 2021. Using a Low Code Development Environment to Teach the Agile Methodology. In *Agile Processes in Software Engineering and Extreme Programming*. Springer International Publishing, 191–199.

[25] Ricardo Martins, Filipe Caldeira, Filipe Sá, Maryam Abbasi, and Pedro Martins. 2020. An overview on how to develop a low-code application using OutSystems. In *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. IEEE, 395–401.

[26] Jose Carlos Metrôlho, Fernando Reinaldo Ribeiro, and Pedro Passão. 2020. Teaching Agile Software Engineering Practices Using Scrum and a Low-Code Development Platform – A Case Study. In *15th Int. Confe. on Software Engineering Advances (ICSEA 2020)*. IARIA, 160–165.

[27] Jose Carlos Metrôlho, R. Araújo, F. Ribeiro, and N. Castela. 2019. An approach using a low-code platform for retraining professionals to ICT. In *EDULEARN19 Proc.* IATED, 7200–7207. https://doi.org/10.21125/edulearn.2019.1719

[28] Lionel Mew and Daniela Field. 2018. A Case Study on Using the Mendix Low CodePlatform to support a Project Management Course. In *EDSIG Conference on Information Systems and Computing Education (EDSIGCON)*. ISCAP.

[29] mgm technology partners GmbH. 2020. *Low Code und Co-Innovation fuer individuelle Enterprise Software.* https://a12.mgm-tp.com/download-whitepaper-a12-enterprise-low-code.html

[30] mgm technology partners GmbH. 2021. *Widget Showcase.* https://a12.mgm-tp.com/showcase/#/

[31] Judith Michael, Dominik Bork, Manuel Wimmer, and Heinrich C. Mayr. 2024. Quo Vadis Modeling? Findings of a Community Survey, an Ad-hoc Bibliometric Analysis, and Expert Interviews on Data, Process, and Software Modeling. *Journal Software and Systems Modeling (SoSyM)* 23, 1 (2024), 7–28. https://doi.org/10.1007/s10270-023-01128-y

[32] Judith Michael, Bernhard Rumpe, and Simon Varga. 2020. Human Behavior, Goals and Model-Driven Software Engineering for Assistive Systems. In *Enterprise Modeling and Information Systems Architectures (EMSIA)*, Vol. 2628. CEUR-WS.org.

[33] Judith Michael and Volodymyr Shekhovtsov. 2024. A Model-Based Reference Architecture for Complex Assistive Systems and its Application. *Journal Software and Systems Modeling (SoSyM)* (2024).

[34] Lukas Netz and Judith Michael. 2024. LCPD usage in Education, public survey results. https://github.com/Lukas-Netz/LCDPs-in-education-survey-data

[35] Nintex. 2024. *Nintex - New ways to fast-forward your workflows.* https://www.nintex.com/

[36] Jose Ignacio Panach, Óscar Dieste, Beatriz Marín, Sergio España, Sira Vegas, Óscar Pastor, and Natalia Juristo. 2021. Evaluating Model-Driven Development Claims with Respect to Quality: A Family of Experiments. *IEEE Transactions on Software Engineering* 47, 1 (2021), 130–145. https://doi.org/10.1109/TSE.2018.2884706

[37] Pega. 2021. *Pega Platform.* https://www.pega.com/products/platform

[38] Laura Poe and Lionel Mew. 2019. Implementing Agile as an Instructional Methodology for Low-Code Software Development Courses.

[39] Clay Richardson and John R. Rymer. 2014. New Development Platforms Emerge For Customer-Facing Applications. (2014).

[40] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *46th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA)*. 171–178. https://doi.org/10.1109/SEAA51224.2020.00036

[41] Raquel Sanchis, Oscar García-Perales, Francisco Fraile, and Raul Poler. 2020. Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences* 10, 1 (2020). https://doi.org/10.3390/app10010012

[42] Massimo Tisi, Jean-Marie Mottu, Dimitrios Kolovos, Juan De Lara, Esther Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. 2019. Lowcomote: Training the next generation of experts in scalable low-code engineering platforms. In *STAF 2019*.

[43] Shouhang Wang and Hai Wang. 2021. A Teaching Module of No-Code Business App Development. *Journal of Information Systems Education* 32, 1 (2021), 1–8.

[44] Robert Waszkowski. 2019. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* 52, 10 (2019), 376–381. https://doi.org/10.1016/j.ifacol.2019.10.060