An Interactive Learning Method to Engage Students in Modeling

Stephan Krusche
krusche@in.tum.de
Technical University of Munich
Munich, Germany

Nadine von Frankenberg
nadine.frankenberg@in.tum.de
Technical University of Munich
Munich, Germany

Lara Marie Reimer
laramarie.reimer@tum.de
Technical University of Munich
Munich, Germany

Bernd Bruegge
bruegge@in.tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

Modeling is an important skill in software engineering. However, it is often not tangible for students and not appreciated. Students prefer coding because they receive immediate feedback from the compiler. Engaging students in modeling is difficult, especially in large introductory courses.

We have developed an interactive learning method for modeling which is based on an easy to use online editor. Students learn modeling in guided tutorials in the lecture right after the theory is introduced and deepen their modeling skills in group work and homework exercises. This learning method was applied in a large introductory course with more than 1000 students.

An empirical evaluation of the method demonstrated that the students' learning outcome in modeling improved significantly by up to 87 %. Students are motivated to use models in their future projects and understand how to approach problems with models. The use of interactive models in programming exercises improves their understanding of the taught concepts.

CCS CONCEPTS

• **Social and professional topics → Software engineering education**; • **Applied computing → Interactive learning environments**; **Learning management systems**.

KEYWORDS

Software Engineering, Education, Learning Management System, Online Editor, Modeling, Learning Success, Interactive

Preprint. To appear at ICSE 2020.

1 INTRODUCTION

Software engineering requires practical application of knowledge [6, 10, 34]. Modeling a system in the Unified Modeling Language (UML) is an important practical skill to facilitate communication between software engineers. Students typically get in touch with UML modeling in undergraduate courses in university. While examples and exercises play a central role in the early phases of cognitive skill acquisition [37], it is time consuming for instructors to create and assess modeling exercises that stimulate all cognitive skills, including creativity. Carefully developed and integrated examples improve the learning outcome [35, 36]. Providing individual feedback and allowing students to improve their knowledge through formative assessments are essential elements in learning [14, 15], so that students can improve their skills.

However, with the rising numbers of students in introductory courses, it is nearly impossible for instructors to teach the creative aspects of modeling and to provide individual feedback. Courses with hundreds of students create enormous efforts for instructors, especially in the correction of exercises and exams, and make it impossible to interact with each student on an individual level [26]. Modeling is a creative task where multiple solutions are possible and it is difficult to judge immediately whether a solution is correct or not [19, 24, 25]. It is not feasible to define all possible correct solutions for a modeling exercise. The choice of the modeling tool has many implications. The use of existing modeling tools such as Visual Paradigm, Gliffy, or draw.io overwhelms and demotivates students due to their complexity. Assessing models with these tools is not possible.

To overcome these problems, we designed an interactive learning method, where students start modeling in an easy to use online editor that focuses on learnability. In this online editor, students receive individual feedback directly next to the model elements in order to avoid media breaks. Students use modeling techniques such as UML to abstract ideas, reduce complexity, improve communication, and to solve concrete problems. The instructor of the course introduces different types of UML diagrams when they are first needed in the software lifecycle.

Through guided tutorials and in-class exercises, students learn the modeling notation of the respective UML diagram type in the lecture. They further practice modeling in tutor exercise sessions. Homework enables students to deepen their modeling skills in self-study. The presentation of their own modeling solution in the accompanying tutor exercise sessions improves communication skills, which are essential in software engineering. Individual feedback on homework allows students to measure their learning progress and improve their skills further. In this paper, we investigate the following hypotheses:

H1 **Learning success**: Interactive learning methods improve the learning success in modeling.
H2 **Engagement**: Integrating modeling throughout the software lifecycle increases student engagement in modeling.
H3 **Understanding**: Interactive models in programming exercises improve students' understanding of the concepts taught.

The remainder of the paper is structured as follows: Section 2 describes related work in relation to relevant learning concepts. Section 3 describes the interactive learning method and the tool support in detail. In Section 4, we present a large software engineering introduction course with more than 1000 students in which we used the interactive learning method. Section 5 outlines the evaluation and its results consisting of an online questionnaire, data analysis and a quasi experiment. Section 6 discusses the advantages and disadvantages of the proposed learning method. Section 7 concludes the paper and proposes future work.

2 RELATED WORK

Software engineering is an engaging, interactive, and collaborative activity [38]. In the educational sector, creating engaging and interactive curricula is an important topic. Content delivery and content exercise is often divided into lectures and exercises which can lead to knowledge gaps. This concept is described by Ebbinghaus's forgetting curve which follows the psychological proposition "... who learns quickly also forgets quickly" [12] and illustrates the knowledge retention rate over time [31]. The forgetting curve implies that after learning new information, within the first 24 hours, there is a retention loss of 40 % to 60 % – if this information is not practiced in short cycles.

Several pedagogical concepts of learning aim at closing this gap of content delivery and practice. Common concepts include: blended learning, a combination of E-learning with the traditional lecturing style that offers the course content through multiple delivery channels [4]; experiential learning, a methodology where students learn from experience [17]; active learning where students participate actively in all learning activities, rather than solely listening passively to a lecturer; and interactive learning, which is based on active and blended learning and further engages students in interactive activities using technology [8, 22].

Engaging students throughout the whole learning process has proved to improve the retention rate. Think-Pair-Share (TPS) is an approach where students work on a problem first individually, then in small groups, and eventually with the whole class [28]. In a study that involved a large introductory programming course, Kothiyal et. al found that a TPS approach yielded 83 % of student engagement [18]. Krusche et. al propose to introduce multiple short iterations between teaching and exercising concepts by combining lectures and exercises into interactive classes [21, 23], and report similar results as Kothiyal, an average of 80 % of student engagement throughout the semester.

Interactive learning concepts involve the direct hands-on application of knowledge, which is important for learning software engineering concepts [34]. The aforementioned concepts are particularly relevant for teaching modeling to students, considering modeling being an informal and creative activity in the software engineering process [7, 11, 19], rather than a concept that should be learned by heart. Some methodologies following this concept focus on collaborative learning environments for UML modeling [1, 9] which may help students to lower the entrance barrier to modeling. Others target the quality assessment and correctness of models [24, 27, 29, 30]. Few approaches propose tools for assessing models or offer scientific evidence on using such tools in practice.

3 LEARNING METHOD

Engaging students in modeling can be challenging. In this section, we describe an interactive learning method that integrates modeling into software lifecycle activities and guides students through their modeling experience.

The learning method is based on active learning [5] and combines lectures and exercises into small iterations to overcome the artificial separation of theory and practice often applied at universities. Instead, instructors introduce small chunks of theory and allow students to exercise them directly. Each theory-exercise cycle consists of five steps [22]:

Theory: The instructor explains a new modeling concept, e.g., a diagram type or modeling technique, and describes the theory behind while students listen and try to understand it.

Example: The instructor shows examples. Students relate the learned modeling concepts and techniques to a concrete situation.

In-class exercise: Students apply the concept in an exercise, e.g., a guided tutorial, and submit their solution.

Feedback: Students receive individual feedback on their own solutions. The instructor provides an example solution, e.g., within the guided tutorial, and explains it to the students to prevent misconceptions [16]. The instructor can also show exemplary student solutions and can discuss their strengths and weaknesses.

Reflection: The instructor facilitates a discussion about the theory and the exercise so that the students reflect on their first experience with the new modeling concept or modeling technique. This can be done, e.g., by discussing best practices, repeating advantages of a technique, or showing how the exercise instantiates the abstract concept.

3.1 Deepening the Learning Content

In addition to in-class exercises, the learning method includes group work exercises and homework. Small tutor exercise sessions with 15-20 students are supposed to deepen the understanding of the concepts explained in the lecture by means of suitable group or team exercises. Students experience the application of the taught concepts and methods with the help of manageable problems in the different phases of software engineering.

Homework exercises enable students to deepen their knowledge in self-study. The teaching concept motivates the students to participate in homework exercises by providing individual feedback and by granting exercise points that can be used to improve the final grade of the course. Individual feedback on homework allows students to measure learning progress and improve their skills.

Students present their solution to group work and homework exercises in the tutor exercise sessions. The presentation of the solution improves communication skills, which are essential in software engineering.

3.2 Tool Support with Artemis and Apollon

Artemis is an exercise system with individual feedback that supports interactive learning and is scalable to large courses [20]. It is open source[1] and used by multiple universities and courses. Artemis integrates an online modeling editor Apollon that is open source[2] and available as standalone and free web application[3]. Apollon supports seven UML diagrams: class diagrams, object diagrams, activity diagrams, use case diagrams, communication diagrams, component diagrams and deployment diagrams. It is lightweight and easy to use to lower the entrance barrier of digital modeling. It focuses on the learning experience of students. Figure 1 shows an example of a UML communication diagram. Students drag the model elements from the right into the diagram and double-click on an element to edit it in a small pop-up. They can drag and drop relationships (e.g. control flow) between model elements.

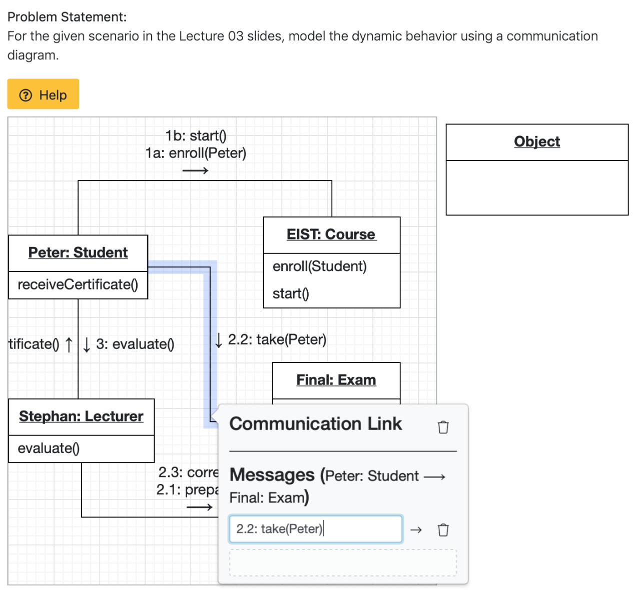
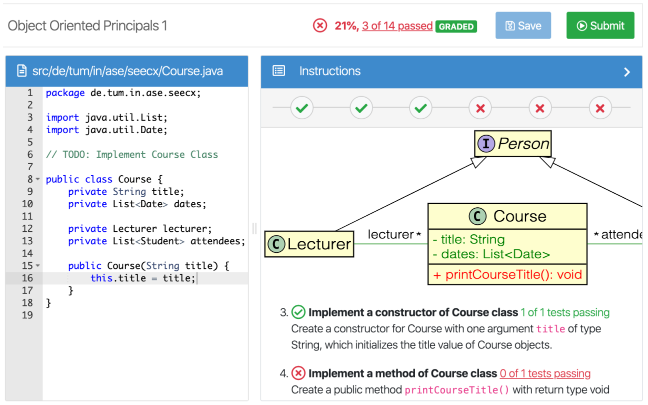

Problem Statement:
For the given scenario in the Lecture 03 slides, model the dynamic behavior using a communication diagram.

Figure 1: The online modeling editor Apollon is integrated into Artemis and supports the easy creation and assessment of digital models.

Instructors and teaching assistants assess models and provide feedback directly in Apollon. They double click on a model element and assess it in a popup with a score in points and with additional feedback comments to explain why a model element is correct or wrong. In addition, they can provide general feedback about the whole model or missing elements. Students can see this feedback directly in place next to the model elements and learn from it.

[1]https://github.com/ls1intum/Artemis
[2]https://github.com/ls1intum/Apollon
[3]https://apollon.ase.in.tum.de

Artemis includes an online code editor with interactive and dynamic exercise instructions on the right [20]. Figure 2 shows a screenshot. Interactive instructions change their color depending on the progress of students. Already completed tasks and correctly implemented model elements are marked in green, incomplete tasks and not yet implemented model elements are marked in red. This helps students to identify which parts of the exercise they have already solved correctly and improves the understanding of the source code on the model level. When they submit their current solution, the interactive instructions update dynamically.


Object Oriented Principals 1
21%, 3 of 14 passed GRADED
Save
Submit
src/de/tum/in/ase/seecx/Course.java
Instructions
Person
Course
Lecturer
lecturer *
* attende
- title: String
- dates: List<Date>
+ printCourseTitle(): void

```java
package de.tum.in.ase.seecx;

import java.util.List;
import java.util.Date;

// TODO: Implement Course Class

public class Course {
    private String title;
    private List<Date> dates;

    private Lecturer lecturer;
    private List<Student> attendees;

    public Course(String title) {
        this.title = title;
    }
}

```

3. ✓ **Implement a constructor of Course class** 1 of 1 tests passing
Create a constructor for Course with one argument `title` of type String, which initializes the title value of Course objects.

4. ✗ **Implement a method of Course class** 0 of 1 tests passing
Create a public method `printCourseTitle()` with return type void

Figure 2: Online code editor with interactive instructions on the right side which include an interactive UML class diagram that changes the color from red to green after successfully completing the corresponding programming task.

4 COURSE

This section describes the undergraduate software engineering course SE1[4] at the Technical University of Munich. The instructors of SE1 introduce students to UML modeling using the interactive learning method described in Section 3.

The learning objectives of SE1 are to familiarize students with relevant concepts, workflows, and methods of software engineering and to apply them in all phases of software engineering projects. This includes analyzing and evaluating problems, e.g., modeling the problem, reusing classes and components, and testing the software. With respect to UML, students learn to communicate using models. They learn how and when to apply which model. They understand the relationship between modeling and programming and learn to abstract. Students learn to model and implement concrete problems in software engineering, for instance with the help of design patterns.

SE1 is a mandatory Bachelor's course offered in the second semester for a heterogeneous group of students from the fields of computer science, business informatics, and business, as well as students from other fields. A prerequisite of the course is that the students have basic programming experience, such as having successfully completed an introductory course in computer science (e.g., CS1). Course instructors use constructive alignment [2] to

[4]The course is called "Introduction to Software Engineering"

align teaching and assessment with the course objectives. For each lecture, a set of learning goals is defined based on the six cognitive skills in Bloom's taxonomy [3]. The course focuses in particular on higher cognitive skills so that students learn to apply the concepts in concrete situations. Students cannot pass the course by simply memorizing the course material.

4.1 Organization

One lecturer and two exercise instructors organize and teach the course with the help of around 45 student tutors. The tutors are bachelor and master students who successfully completed the course in previous years. The course takes place in the summer semester over 12 weeks. Table 1 shows the course content together with the UML models that are taught in the respective lecture. Around 1600 students register for the course. There is no lecture hall with enough seats for all students. The course uses a live stream and broadcasts the lecture to two additional overflow lecture halls. Most students either participate actively in the main lecture hall or watch the live stream at home. Therefore, the overflow lecture halls are closed after a few weeks. All students (within the main lecture hall, in overflow lecture halls and in the live stream) can ask questions using Slack[5]. Tutors answer these questions directly or pass on a question to the lecturer to repeat and answer it for all students.

Week	Content	UML Model
1	Introduction	Class
2	Model-based SE	Use Case, Class
3	Requirements Analysis	Object, Communication
4	System Design I	Component, Deployment
5	System Design II	Component, Deployment
6	Object Design I	Class
7	Object Design II	Class
8	Model Transformation and Refactoring	State Chart
9	Software Lifecycle Modeling	Activity
10	Software Configuration Management	Activity
11	Testing	Class
12	Project Management	Class, Activity

Table 1: Course Schedule: SE1 lasts 12 weeks. Each lecture includes specific UML models.

4.2 Design

Large courses present the challenge of keeping students motivated throughout the semester (without, e.g., enforcing mandatory attendance). Students are easily distracted by off-topic conversations with other students or social media, and stop paying attention to the lecture. To deal with such situations, the course includes interactive elements to activate the students and to keep students engaged throughout the course. The interactive components include in-class exercises, in-class quizzes, and group exercise sessions.

The study program of the university does not allow to include weekly assignments in the calculation of the final grade. Therefore, the course uses a bonus system that motivates students to participate in the course and its exercises: students can earn bonus points for completing in-class and homework exercises successfully. They need to present their homework twice in tutor exercise sessions to get the bonus applied.

If they pass the final exam, their exercise points are mapped to exam points that are then added to their final exam score to improve it. The German grading system consists of marks between 1.0 (similar to *A* in the US grading system) and 5.0 (similar to *F* in the US grading system), with 1.0 being the highest grade and 4.0 (similar to *D* in the US grading system) being the pass grade. For instance, if students score 30 % of the bonus points, they receive additional 3.0 points on top of their exam score, which improves their final grade by 0.3. If they score 100 % of the bonus points, they can receive a total bonus of 1.0, for instance, they can improve from a 2.3 to a 1.3. Students have reported that this increases their motivation to actively participate in the exercise system.

4.3 Exercises

The course includes quizzes, programming, modeling, and text exercises. In-class exercises include quizzes, to recapture previously learned content. They also include programming and modeling exercises as guided tutorials. Tutors help with student questions and problems during the in-class exercises. Group exercises mainly encompass modeling exercises, but also small programming exercises and text exercises that the students work out together during their group exercise sessions in small teams. Homework assignments include modeling, programming, and text exercises and enable students to deepen their knowledge in self-study.

4.3.1 Modeling Exercises. Students model a solution to concrete problems using UML. Modeling exercises stimulate higher cognitive skills and force students to analyze, evaluate and create. Apollon supports UML class, object, activity, use case, communication, component, and deployment diagrams. As shown in Table 1, each lecture includes different UML model types, aligned with the taught content. Figure 3 shows an example for a modeling exercise. Students drag and drop the model elements into the canvas, can add attributes, methods, and define associations between them. The advantage of Apollon is that students cannot use a UML element other than the ones specified for the specific model type.

4.3.2 Quiz Exercises. Students repeat already learned content during lectures and test their knowledge. They stimulate lower cognitive skills such as remembering and understanding the concepts. A quiz question can either be a multiple choice (MC) question, a drag and drop (DnD) question, or a short answer question. For questions related to modeling, the quizzes include MC and DnD questions where students drag elements to predefined spots on the canvas.

4.3.3 Programming Exercises. Students learn to make connections and see differences between models and their implementation in programming exercises. This stimulates their cognitive skills and students learn to apply the knowledge when implementing source code. A UML class diagram, e.g., represents the general structure of the source code and can be used as interactive problem statement in Artemis. Red model elements indicate that they are not implemented correctly, whereas green elements indicate correctly

[5] Slack is a cloud-based instant messaging platform: https://slack.com

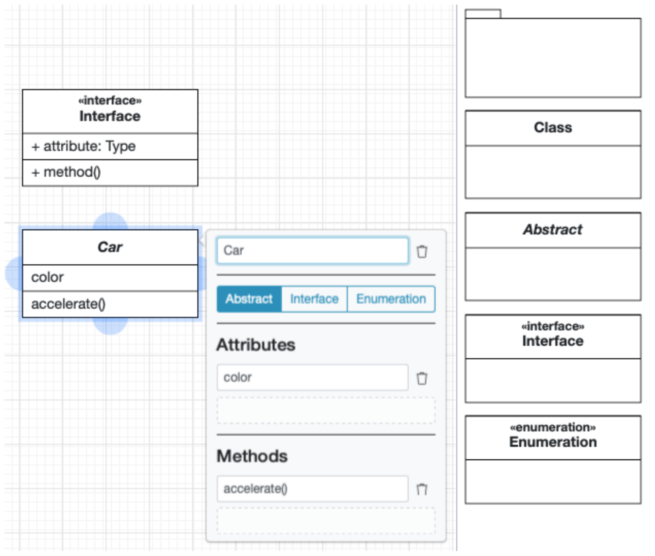


Figure 3: An example for a modeling exercise in Apollon.

implemented ones. This further helps students to understand what UML models should contain and what should be left out.

4.3.4 Text Exercises. Students need to answer questions about the learned concepts by writing open text responses. They, e.g., need to explain similarities and differences between design patterns in their own words and describe concrete situations how design patterns can be used. These exercises stimulate analysis and evaluation skills.

4.4 Communication

The course uses Slack as communication tool to facilitate discussions between students and teaching staff. Using instant messaging lowers the entrance barrier for students to ask questions, because it feels familiar to communicate (in reference to, e.g., social media chats) and students ask more questions if they notice that other students do the same. Students can communicate with each other and send direct messages to tutors and instructors in case they have a question or require help during the lectures and at any other time. Slack offers the ability to use channels with specific purposes:
#announcements — Instructors post course-wide announcements (students cannot post here), e.g., reminders that a lecture is cancelled on a public holiday
#organization — Questions about the organization of the course
#lecture — Questions regarding the lecture slides
#exercise — Questions regarding the exercises

The instructors further encourage students to answer questions themselves. This increases a 'sense of belonging' (which is hard to achieve in such a large setting), when students communicate with each other, but can also deepen their understanding of a topic, e.g., in discussions that pursue questions. Students receive fast replies, which increases the interactivity. Tutors help to moderate discussions. They ensure a positive atmosphere, reprimand and prevent bullying. They answer questions and point students to previously asked questions, if it has already been asked before.

4.5 Tutor Exercise Sessions

45 tutors hold 80 weekly occurring tutor exercise sessions, each with around 20 students. The main focus for tutors is to activate the students in these sessions, to moderate discussions and to explain the learned concepts again in case the students ask questions. Tutors have the following responsibilities: attend a weekly tutor meeting with the instructors[6], assess exercises and hold one or two tutor exercise sessions per week. Tutors also help with moderating the Slack channels, answer questions on Artemis, help students during in-class exercises, or review slides and exercise content.

In the tutor exercise sessions, students apply the knowledge acquired in the lecture. Each tutor exercise session is structured as follows:

(1) **Review of previous lecture** [5 - 10 min]: students discuss the learning goals, outline, and summary.
(2) **Homework presentation** [30 - 45 min]: students present their solution to homework exercises. Tutors asks questions about the solution, point out typical mistakes and provide additional feedback.
(3) **Group work** [30 - 45 min]: Students work on predefined group exercises in groups (3-6 students).
(4) **Discussion of next homework** [5 - 10 min]: the new homework exercises are briefly discussed.

The tutor exercise sessions review a specific topic that was covered in the lecture before and prepare the students for the next homework assignment. They help to deepen the understanding of the taught concepts. Group exercises show the application of the learned methods with the help of concrete problems in the different phases of software engineering. Homework assignments deepen the knowledge in self study. Students receive individual feedback on their homework submission, which allows them to measure their learning progress and improve their skills. The presentation of their own solution improves the communication skills of the students, an essential skill in software engineering.

For instance, in lecture *Object Design II*, the course covered the Strategy Design Pattern [13] by means of an example and the general structure. In the corresponding tutor exercise session, there was one group work, where students discussed the pattern's problem, solution, benefits, consequences, etc. In the subsequent group work, the students modeled a real-world example of the strategy pattern as a UML class diagram. This exercise was designed to teach students how to approach a concrete problem, how to analyze it, and how to model this problem. In one homework assignment, the students were given a similar problem: to model different encryption strategies as a UML class diagram, using the strategy pattern. In another assignment in a programming exercise, the students had to implement sorting algorithms using the strategy pattern.

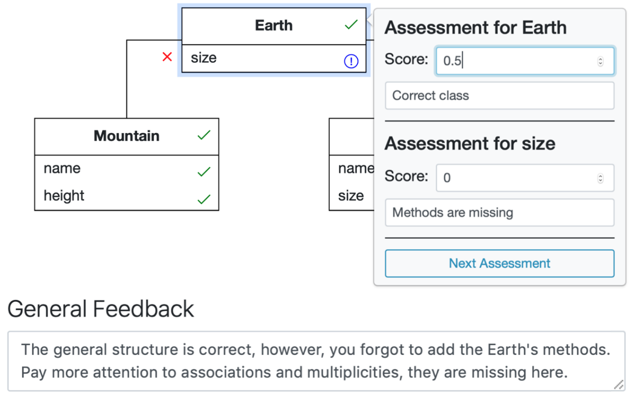
4.6 Grading

While programming and quiz exercises are automatically evaluated, modeling and text submissions are graded manually. Each tutor grades about 25 submissions per exercise per week. Artemis offers a double-blind grading system, which opts for less bias while grading. Every week on Monday at noon, the homework is published. The

[6]Instructors discuss issues and present the next group work and homework.

students then have one week to create and upload their solutions. In the following week, students present their homework in the tutor exercise sessions. Tutors use example solutions and detailed grading criteria to assess the students' submissions and provide individual feedback. In the grading criteria, instructors point out that multiple solutions to modeling exercises can be correct.

When students are given sample solutions, they often do not think about their own solutions, but tend to take the sample solution as the single truth. To encourage self-reflection and revision, the example solutions are not distributed to the students. The feedback students receive about their solutions is crucial for them to understand how they can improve. As shown in Figure 4, the feedback for modeling exercises is comprised of the (1) points they receive for one concrete element, (2) feedback for this element, and (3) general feedback regarding the whole model.



The general structure is correct, however, you forgot to add the Earth's methods. Pay more attention to associations and multiplicities, they are missing here.

Figure 4: An example for a modeling exercise with individual feedback.

5 EVALUATION

This section presents the evaluation of the interactive learning method in the SE1 course in 2019 based on the hypotheses in Section 1. We describe the research method, present the results and discuss the findings and limitations.

5.1 Research Method

In order to test our hypotheses, we applied the following three research methods:

(1) **Online Questionnaire**: we created an online questionnaire and asked all participants of the SE1 course in 2019 to participate.
(2) **Data Analysis**: we analyzed the participation and exercise performance of the students in the modeling exercises in the SE1 course in 2019.
(3) **Quasi Experiment**: we compared the results of modeling tasks in the SE1 exams from 2018 and 2019. In 2018, the SE1 course did not include the interactive learning method for modeling exercises and therefore serves as the control group.

5.1.1 Online Questionnaire. All participants of the course SE1 in 2019 were asked to complete a questionnaire about their experience with the modeling exercises. The questionnaire consisted of four main parts, each containing several questions: (1) demographic information, in particular field of study, current semester, and the student's achieved prerequisites for SE1; (2) previous modeling and programming experience (before taking SE1); (3) participation in SE1; (4) modeling in SE1, including motivation, the interactive learning method, and the tools being used.

5.1.2 Data Analysis. We analyzed the results of all modeling exercises in SE1 in 2019. This includes a total of 17 modeling exercises; 9 of which were conducted as in-class exercises, the other 8 as homework. Each of the modeling exercises was assigned a difficulty level[7] as well as a score. To analyze each modeling exercise, we have created a dataset that contains all participating students as well as their individual scores per exercise. Based on the scores of the participating students, we calculated the average success rate in % of the total score.

5.1.3 Quasi Experiment. We carried out a quasi experiment with post-testing of two student groups, i.e., students who took SE1 in 2018 and students who took SE1 in 2019, by comparing their scores in the modeling tasks of the final exam. Both course instances in 2018 and 2019 had the same learning goals, the same course schedule with the same content and the same exercise structure except for modeling exercises: In 2018, SE1 did not use the interactive learning method and instead relied on practicing modeling only in homework. In 2019, SE1 used the interactive learning method and introduced in-class modeling exercises. In terms of the quasi experiment, the interactive learning method is the intervention. Apart from that, there were no substantial differences in other variables.

The control group is comprised of the 2018 students, the experimental group of the 2019 students. We did not execute a pre-test. Our assumption was that students from both groups had similar knowledge regarding modeling before taking part in the SE1 course, mainly because the majority of both student groups was comprised of second-semester bachelor students, with both groups following the same curriculum. In both years, the course was attended by over 1000 students, so that a normal distribution of the results can be assumed. Both exams included five similar modeling tasks:

(1) **Functional model**: Create a UML use case diagram based on a given problem statement (easy)
(2) **Structural model**: Create an analysis object model using a UML class diagram based on a given problem statement (medium)
(3) **Dynamic model**: Create an UML activity diagram (2018) / UML communication diagram (2019) based on a given problem statement (medium)
(4) **Architecture**: Create a UML communication diagram of an architectural style (2018, medium) / model the architecture based on a given problem statement using a UML component diagram (2019, hard)
(5) **Model refactoring**: Analyze an existing model, propose a model refactoring and explain the reasoning (easy)

In the post test, we compared these five modeling tasks in two-sample one-tailed t-tests to evaluate, whether the 2019 students performed significantly better than the 2018 students. For all model tasks, the null hypothesis H_0 is that the 2019 students performed

[7]Possible difficulty levels: **E** = easy, **M** = medium, or **H** = hard

less or equal as compared to the 2018 group with a significance level of $\alpha = 0.01$. H_1 hypothesizes that the 2019 results are better than the results from 2018. 1128 students completed the exam in 2018, 1225 completed the exam in 2019. To make the results comparable (two tasks differed by one point) we calculated the mean and standard deviation as relative values.

5.2 Results

This subsection shows the results of the three research methods.

5.2.1 Online Questionnaire. In total, 954 students participated in the online questionnaire (response rate: 68 %). 90 % of the students are enrolled in a Bachelor's program, while 10 % of the participants are enrolled in a Master's program. 69 % of the students take SE1 in their second semester, followed by 18 % forth-semester students.

The first question (Q1) refers to the experience the student's had with UML modeling before taking the SE1 course. Figure 5 depicts the answer distribution of Q1. 50 % of the participants stated that they have "somewhat experience", which means that they have modeled using UML once in a previous course, followed by 29 % of students students that stated that they have "little experience", which means that they had heard about UML models before. 17 % had no experience at all, only 4 % stated that they model regularly.

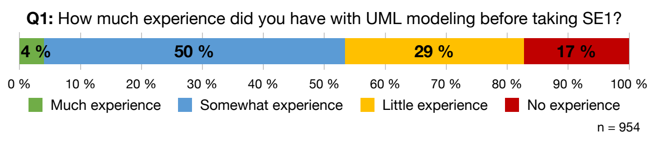


Figure 5: Limited modeling experience before SE1.

Q2 refers to the frequency of participation in a tutor exercise session. 52 % of the students stated that they always attend the tutor exercise sessions, while 27 % stated that they visit them very often. 12 % participate sometimes, 7 % rarely attend and 2 % never participate. Figure 6 depicts the distribution of answers.

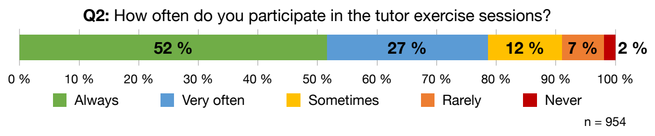


Figure 6: Regular participation in tutor exercise sessions.

In Q3, students were asked how often they submit their solutions to the modeling exercises. The majority, 74 %, stated that they submit always, 10 % submit very often. 9 % submit sometimes, while 5 % submit rarely, and 2 % have never submitted any modeling exercise. The results are shown in Figure 7.

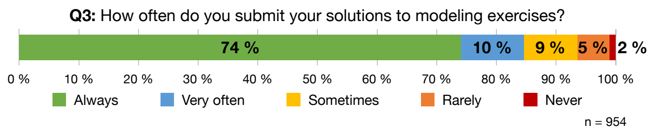


Figure 7: Regular submissions of modeling exercises.

Q4 stated that the interactive models used in Artemis programming exercises have helped the students to solve the exercises. The rating was done by using a 5-point Likert scale as shown in Figure 8. 54 % strongly agree with the statement, 36 % agree. 6 % have a neutral opinion on the statement, whereas 3 % disagree and 1 % of the participants strongly disagree.

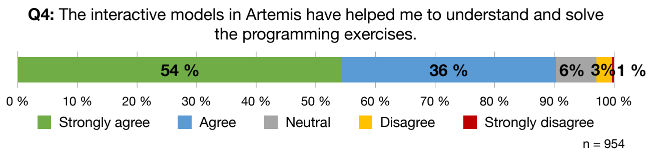


Figure 8: Interactive models in programming exercises.

Q5 asked whether students would use models in their future projects using a 5-point Likert scale (shown in Figure 9). 24 % of the students stated that they strongly agree, 51 % stated that they agree with this statement. 20 % have a neutral opinion. 4 % of the students disagree and 1 % strongly disagree.

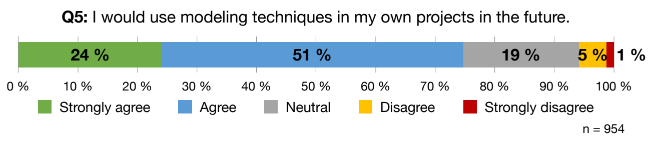


Figure 9: Use of modeling in future projects.

Q6 analyzed to which extent the students considered the different exercise types as helpful (shown in Figure 10). Homework exercises were rated most helpful for deepening and understanding modeling after the theory was introduced (37 % strongly agree and 47 % agree), followed by in-class exercises (28 % strongly agree, 49 % agree). Group works were considered least helpful, with 13 % of students strongly agreeing to the statement, 36 % agreeing, 31 % neutral opinions, 16 % disagreeing and 4 % strongly disagreeing.

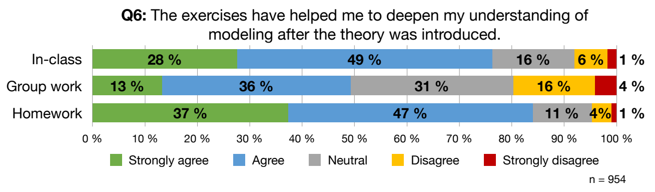


Figure 10: Helpfulness of exercise types to deepen the understanding of modeling.

Q7 asked the participants to state their opinion on different statements regarding the modeling exercises and concepts using a 5-point Likert scale. Figure 11 shows the results. The exercises and concepts especially helped the students to understand why to use models (31 % strongly agree, 53 % agree) and improve their modeling skills (32 % strongly agree and 53 % agree). 72 % state that modeling helped them to understand how to approach problems in software engineering.

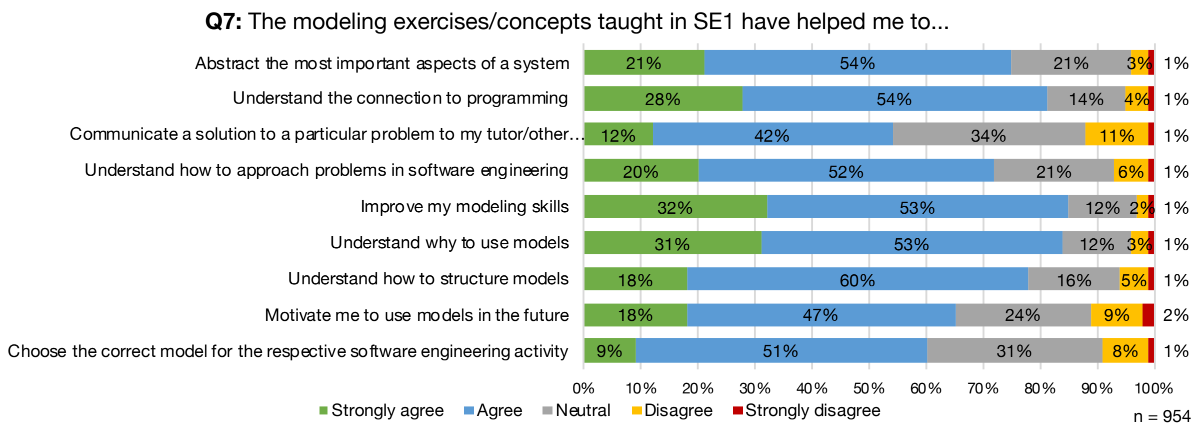


Figure 11: Helpfulness of modeling exercises in various aspects.

5.2.2 *Data Analysis.* In the second part of the evaluation, we analyzed data from the Artemis system regarding the modeling exercises conducted during the SE1 2019 course. SE1 included 17 modeling exercises, 9 of them were guided tutorials done in the lecture, 8 of them were executed by the students on their own as homework. The results are summarized in Table 2.

#	Exercise	Pts	Difficulty	Participations	Avg. Score
1	L1 Modeling Tutorial	4	E	1147	97 %
2	H1 Use Case Model	4	E	1116	55 %
3	H2 Analysis Object Model	6	E	1090	75 %
4	L2 Communication Diagram	6	E	1136	88 %
5	H3 Communication Diagram	7	M	992	54 %
6	L3 Model View Controller	4	H	1070	99 %
7	L4 Component Diagram	3	M	1049	98 %
8	H4 Choose a Design Pattern	6	M	899	83 %
9	H5 Strategy Pattern	5	M	886	91 %
10	L5 Model Refactoring	2	E	950	90 %
11	L6 Java to UML	2	E	898	67 %
12	H6 Tables to a Model	4	M	851	74 %
13	L7 Scrum as Activity Diagram	5	M	860	89 %
14	H7 Activity Diagram	8	M	884	82 %
15	H8 Build & Release Management Workflow	10	M	851	95 %
16	L8 Functional Model	4	E	819	62 %
17	L9 Analysis Object Model	6	M	810	74 %

Table 2: Students actively participate throughout the course. They score more points in in-class exercises (L) than in homework exercises (H) on average.

The table depicts the different exercises, including a unique identifier, e.g. L1, where "L" represents that the exercise was conducted during a lecture. Exercises conducted as Homework are marked with a leading "H". All exercises are assigned a difficulty, there were 7 easy exercises, 9 medium ones and 1 hard one in the area of software architectures. Each exercise in the table has a certain amount of points (between 2 and 10) that the students can achieve, as well as the number of participations and the average score achieved in %. On average, the guided exercises performed in-class received a higher average score (85 %) than the homework exercises, that the students had to solve on their own (76 %).

5.2.3 *Quasi Experiment.* We compared the exam results SE1 in 2018 against the results of SE1 in 2019. Both exams covered the same exercise types about functional, structural, dynamic, and architecture models as well as refactoring of an existing model. We calculated the average score the students reached in the exercises and compared them against each other. The results are shown in Figure 12.

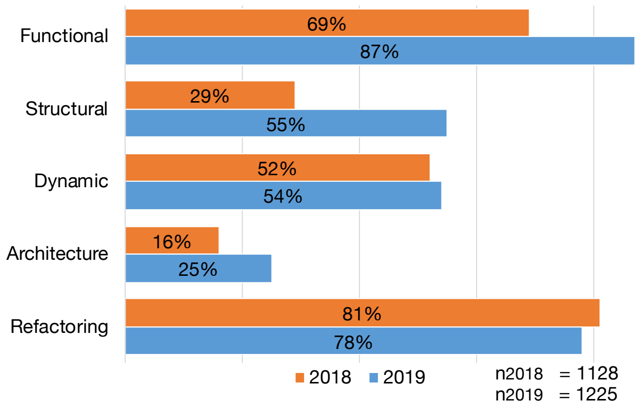


Figure 12: Students scored significantly better in 3 modeling tasks in the 2019 exam than in the 2018 exam.

Except for the exercise about refactoring, where students received 4 % fewer points on average in 2019 (78 %) than in 2018 (81 %), the students performed better in the SE1 2019 exam than in the 2018 exam in the analyzed modeling tasks. For the functional model, the average score is at 87 %, which is 26 % higher than in the 2018 exam, where students received 69 % of available points on average. In the structural model, students received 55 % of the points on average, which is 87 % more than in 2018, where they received 29 % on average. The results for the dynamic model are 4 % higher (54 %) than in 2018 (52 %). The average score of the architecture exercise is 55 % higher in 2019 with 25 % compared to 16 % in 2018.

To evaluate the significance of the results, we performed a two-sample one-tailed t-test with a significance level of $\alpha = 0.01$. The results of the t-tests per exercise are depicted in Table 3. If the p-value for the exercise was below alpha, we rejected H_0 and considered the results from 2019 as significantly better than 2018. For the functional and structural models, we calculated a p-value of 2,2e^{-16} and for the architecture model a p-value of 6,4e^{-15}, which means, that students were significantly better in the 2019 exam for the same model type in 2018. For the dynamic model, we calculated a p-value of 0,09, for the refactoring, the p-value was 0,99. This means, that the 2019 results were not significantly better than in 2018.

Model	$\overline{x}_{2018}$	$\overline{x}_{2019}$	σ_{2018}	σ_{2019}	p
Functional	0,689	0,867	0,334	0,237	2,2e^{-16}
Structural	0,293	0,549	0,265	0,310	2,2e^{-16}
Dynamic	0,519	0,538	0,327	0,365	0,09
Architecture	0,161	0,249	0,273	0,278	6,4e^{-15}
Refactoring	0,812	0,776	0,281	0,208	0,99

$n_{2018} = 1128$, $n_{2019} = 1225$ $\alpha = 0,01$

Table 3: T-tests of the SE1 2018 and SE1 2019 underline that the 2019 students were significantly better for functional, structural and architecture models.

5.3 Findings

SE1 serves as introduction to modeling for most students in the computer science curriculum. The modeling experience of the students before the course is low.

The results show that the interactive learning method motivates the students to attend the tutor exercise sessions and submit their modeling exercises. The performance in the modeling exercises indicates that, depending on the individual exercises, the amount of good and successful submissions is high, with an average score of 85 % for guided in-class exercises and 76 % in homework exercises. The main reason for the differences in the success rates between in-class and homework exercises lies in the fact that in-class exercises are usually performed together with the instructor. The instructor explains and performs the exercise together with the students using a projector. The students are able to see the correct steps of creating the model by following the instructions.

The in-class exercises serve as a tutorial and aim at giving the students a first hands-on experience in the specific UML model type. Later on, the students have to apply their new knowledge on their own in the homework. There are no instructions other than the exercise description, so that they have to find a solution without the help of the instructor. As this is usually more difficult and more error-prone than following tutorial steps, the average score is lower. When comparing the examinations of SE1 in the 2019 with the control group in 2018, the results in the modeling exercises were significantly better for the 2019 group, which used the interactive learning method.

Finding 1: Interactive learning improves the learning success in modeling.

SE1 covers different UML models, that are being created throughout the whole software lifecycle. The teaching schedule and modeling exercises are aligned to cover different types of modeling in the corresponding lecture and following tutor exercise sessions and homework exercises. The students get an overview about when to use which UML model. The results show that most participating students submit their modeling exercises every week or at least on a regular basis.

The number of participations between the first (1147 participations) and the last modeling exercises (810 participations) stayed on a high level compared to traditional courses, where usually only about 25 % of the students still actively participate in the last lectures and exercises. The results of the questionnaire show, that students are highly motivated because of the interactive learning method and the use of modeling throughout the whole software engineering lifecycle. Higher motivation causes students to engage more in the lecture.

Finding 2a: Integrating modeling throughout the whole software lifecycle increases student engagement in modeling.

Finding 2b: Interactive learning increases student engagement in modeling.

Most of the participating students in the survey reported that the interactive learning method has helped them to approach different problems in software engineering. They understand the connection between modeling and programming. Especially the usage of the interactive UML class diagrams in the programming exercises helped them to better understand the programming exercises and find the correct solution.

Finding 3: Interactive models in programming exercises improve the students' understanding of the taught concepts.

Most of the students also state they would use modeling techniques in their own projects in the future and that it helped them to better understand how to approach problems in software engineering. This is also emphasized by the success of the students in the 2019 exam compared to their results in the 2018 exam, where the interactive learning method was not used.

5.4 Threats to Validity

Internal validity: The evaluation does not measure all variables that could lead to better exam results. Existing knowledge, motivation, the exact wording of an exam question and other external factors might influence how students perform a modeling task in an exam. The internal validity of the results in the quasi experiment might be limited [32]. However, the online questionnaire and the data analysis support the findings.

External validity: The SE1 course is one specific example of courses, where modeling is taught. It is a mandatory course that is offered to many students, who may differ in their field of studies as well as in their previous experiences. We assume that the interactive learning method can be successfully applied in other software engineering courses and is generalizable. However, other study programs and regulations might make it difficult to adopt the approach.

Construct validity: The validity of the questionnaire might be affected by the wording of the questions or due to the fact that students like the approach of getting feedback, which does not necessarily improve their learning outcome. To limit the influence, we carefully designed the questions, used Likert scales as answer options and multiple researchers reviewed the wording. The measures in the quasi experiment and the data analysis support the findings.

6 DISCUSSION

While individual feedback for modeling exercises improves the students understanding and retention rate in terms of modeling and problem solving, as shown in Section 5.3, assessing each model submission increases the workload. Due to the amount of exercises to be assessed, it is hard to provide meaningful and understandable feedback comments, which can lead to partly incomprehensible and insufficient feedback. We have observed that tutors take more time at the beginning and take less time towards the end. Some tutors lack the required expertise or motivation which can result in inconsistent assessments in terms of fairness and correctness.

With either missing or insufficient feedback comments, students may get confused and would require a more extensive feedback to understand their faults. We have added a "request more feedback" functionality to Artemis, so that students can ask specific questions and another tutor answers them. A lack of feedback may also motivate students to participate more in discussions during the group work exercises, where they can ask their tutor for clarification. Artemis helps to assess student submissions with less bias than traditional methods, because student and tutor names are hidden. A random allocation of assessments also improves the fairness of assessments.

One of our main goals is to teach creativity in modeling which can further enhance problem solving abilities as well as understanding of object-orientated principles. For students new to modeling, this can be difficult. First, students are used to solve programming exercises where the compiler tells them exactly where they made a mistake, in real-time. This allows students to find and fix mistakes easily. Second, students are often used to precise problem statements and assume only one correct solution. They may have trouble in understanding that multiple solutions can be correct, which is the case in modeling. Third, students new to modeling may also lack the confidence in creating, presenting and discussing their own solutions with tutors or peer students. In our learning method, we approach these challenges by instructing tutors to actively encourage discussions and asking students to present their solutions to homework exercises at least twice during the tutor exercise sessions in order to qualify for the bonus system.

There are certain benefits for modeling on paper first, rather than using online editors, regarding syntax, semantic, and aesthetics [33]. Students may become used to using an online editor that already provides the syntax. In the paper-based exam, they would have to remember the syntax themselves and still draw their solutions on paper. Online editors, on the other hand, offer the function to change models easily, e.g., by adding a new element and replacing existing elements. On paper, this can quickly become unreadable and cumbersome.

Extrinsic motivation might also be a factor, because learning may be affected by assessments and bonus points for their work, rather than solely focusing on feedback and learning outcome. In our experience, especially students in their first year can have trouble in differing between learning and receiving feedback. The bonus might motivate those students more to participate in the exercises than the effect of practicing and improving their skills.

7 CONCLUSION

In this paper, we presented an interactive learning method that teaches modeling in an interactive setting, where students learn the problem solving aspects of modeling, can iterate through different types of models, and discuss their results in tutor exercise sessions. We present how Artemis and Apollon support students, instructors, and tutors in learning and teaching modeling throughout the whole software engineering lifecycle. We applied the interactive learning method in a large introductory software engineering course with more than 1000 students. Our empirical evaluation consisting of an online questionnaire, data analysis, and a quasi experiment shows that the interactive learning method improves the learning success of the students significantly and increases their motivation in modeling.

In the future, we want to integrate team projects with realistic problem statements and modeling tasks in our courses that are supported by the interactive learning method. Modeling with real-time synchronization in Apollon would allow students to collaborate on the model creation even in distributed settings. We also want to incorporate peer reviews for modeling exercises in Artemis so that students also have to grade models and provide feedback to other students because evaluating a model further improves the own modeling skills.

REFERENCES

[1] Mohammed Basheri, Liz Burd, and Nilufar Baghaei. 2012. Collaborative software design using multi-touch tables. In *4th International Congress on Engineering Education*. IEEE, 1–5.

[2] John Biggs. 2003. Aligning teaching and assessing to course objectives. *Teaching and learning in higher education: New trends and innovations* 2, 13–17.

[3] Benjamin Bloom, Max Engelhart, Edward Furst, Walker Hill, and David Krathwohl. 1956. Taxonomy of Educational Objectives: The Classification of Educational Goals.

[4] Curtis Bonk and Charles Graham. 2012. *The handbook of blended learning: Global perspectives, local designs*. John Wiley & Sons.

[5] Charles Bonwell and James Eison. 1991. *Active Learning: Creating Excitement in the Classroom*. ASHE-ERIC Higher Education Reports.

[6] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. 2015. Software Engineering Project Courses with Industrial Clients. *ACM Transactions on Computing Education* 15, 4, 17:1–17:31.

[7] Bernd Bruegge, Stephan Krusche, and Martin Wagner. 2012. Teaching Tornado: from communication models to releases. In *Proceedings of the MODELS Educators' Symposium*. ACM, 5–12.

[8] Doug Buehl. 2017. *Classroom strategies for interactive learning*. Stenhouse Publishers.

[9] Weiqin Chen, Roger Heggernes Pedersen, and Øystein Pettersen. 2006. CoLeMo: A collaborative learning environment for UML modelling. *Interactive Learning Environments* 14, 3, 233–249.

[10] Thomas Connolly, Mark Stansfield, and Thomas Hainey. 2007. An application of games-based learning within software engineering. *British Journal of Educational Technology* 38, 3, 416–428.

[11] Dora Dzvonyar, Stephan Krusche, and Lukas Alperowitz. 2014. Real Projects with Informal Models. In *Proceedings of the MODELS Educators Symposium*. 39–45.

[12] Hermann Ebbinghaus. 2013. Memory: A contribution to experimental psychology. *Annals of neurosciences* 20, 4, 155.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design patterns: Abstraction and reuse of object-oriented design. In *European Conference on Object-Oriented Programming*. Springer, 406–431.

[14] Richard Higgins, Peter Hartley, and Alan Skelton. 2002. The conscientious consumer: Reconsidering the role of assessment feedback in student learning. *Studies in higher education* 27, 1, 53–64.

[15] Alastair Irons. 2007. *Enhancing learning through formative assessment and feedback*. Routledge.

[16] Paul Kirschner, John Sweller, and Richard Clark. 2006. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist* 41, 2, 75–86.

[17] Alice Kolb and David Kolb. 2005. Learning styles and learning spaces: Enhancing experiential learning in higher education. *Academy of management learning & education* 4, 2, 193–212.

[18] Aditi Kothiyal, Rwitajit Majumdar, Sahana Murthy, and Sridhar Iyer. 2013. Effect of think-pair-share in a large CS1 class: 83% sustained engagement. In *Proceedings of the 9th annual international conference on International computing education research*. ACM, 137–144.

[19] Stephan Krusche, Bernd Brügge, Irina Camilleri, Kirill Krinkin, Andreas Seitz, and Cecil Wöbker. 2017. Chaordic Learning: A Case Study. In *39th International Conference on Software Engineering: Software Engineering Education and Training*. IEEE, 87–96.

[20] Stephan Krusche and Andreas Seitz. 2018. ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In *Proceedings of the 49th Technical Symposium on Computer Science Education (SIGCSE)*. ACM, 284–289.

[21] Stephan Krusche and Andreas Seitz. 2019. Increasing the Interactivity in Software Engineering MOOCs - A Case Study. In *52nd Hawaii International Conference on System Sciences*. 1–10.

[22] Stephan Krusche, Andreas Seitz, Jürgen Börstler, and Bernd Bruegge. 2017. Interactive learning: Increasing student participation through shorter exercise cycles. In *Proceedings of the 19th Australasian Computing Education Conference*. ACM, 17–26.

[23] Stephan Krusche, Nadine von Frankenberg, and Sami Afifi. 2017. Experiences of a Software Engineering Course based on Interactive Learning. In *Tagungsband des 15. Workshops Software Engineering im Unterricht der Hochschulen*. 32–40.

[24] Christian Lange and Michel Chaudron. 2004. An empirical assessment of completeness in UML designs. In *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering*. 111–121.

[25] Christian Lange, Bart DuBois, Michel Chaudron, and Serge Demeyer. 2006. An experimental investigation of UML modeling conventions. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 27–41.

[26] Harold Leavitt and Bernard Bass. 1964. Organizational psychology. *Annual Review of Psychology* 15, 1, 371–398.

[27] WenQian Liu, Steve Easterbrook, and John Mylopoulos. 2002. Rule-based detection of inconsistency in UML models. In *Workshop on Consistency Problems in UML-Based Software Development*, Vol. 5.

[28] Frank Lyman. 1987. Think-pair-share: An expanding teaching technique. *Maa-Cie Cooperative News* 1, 1, 1–2.

[29] Jacqueline McQuillan and James Power. 2006. On the application of software metrics to UML models. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 217–226.

[30] Kashif Mehmood and Samira Si-Said Cherfi. 2009. Evaluating the Functionality of Conceptual Models. In *ER Workshops*.

[31] Jaap Murre and Joeri Dros. 2015. Replication and analysis of Ebbinghaus' forgetting curve. *PloS one* 10, 7.

[32] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples* (1st ed.). Wiley Publishing.

[33] Doris Schmedding and Anna Vasileva. 2017. Reviews-ein Instrument zur Qualitätsverbesserung von UML-Diagrammen.. In *SEUH*. 8–19.

[34] David Shaffer. 2004. Pedagogical praxis: The professions as models for postindustrial education. *Teachers College Record* 106, 7, 1401–1421.

[35] John Sweller and Graham A. Cooper. 1985. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction* 2, 1, 59–89.

[36] J. Gregory Trafton and Brian J. Reiser. 1993. *Studying Examples and Solving Problems: Contributions to Skill Acquisition*. Technical Report. Naval HCI Research Lab, Washington, DC, USA.

[37] Kurt VanLehn. 1996. Cognitive Skill Acquisition. *Annual Review of Psychology* 47, 513–539.

[38] Jim Whitehead. 2007. Collaboration in Software Engineering: A Roadmap. *FOSE* 7, 214–225.