Aligning the Learning Experience in a Project-Based Course: Lessons Learned from the Redesign of a Programming Lab

Malte Mauritz
TU Dortmund University
Dortmund, Germany
malte.mauritz@tu-dortmund.de

Stefan Naujokat
TU Dortmund University
Dortmund, Germany
stefan.naujokat@tu-dortmund.de

Christian Riest
TU Dortmund University
Dortmund, Germany
christian.riest@tu-dortmund.de

Till Schallau
TU Dortmund University
Dortmund, Germany
till.schallau@tu-dortmund.de

ABSTRACT

In teaching and training the next generation of software engineers, programming labs with students working together in small groups provide the opportunity to obtain hands-on experience for software projects involving multiple developers. However, more than other types of courses, programming labs face some challenges in providing a similar learning outcome for all students. Based on feedback and own experience from various iterations of the programming lab at TU Dortmund University, we identified that the learning experience varies significantly due to heterogeneous prior knowledge, experience levels, and personality traits of both students and tutors.

In this experience report, we present our approach towards aligning the learning experience by applying three different didactic improvements based on well-studied concepts: (1) the idea of *worked-out examples* is transferred to teaching the software development process by providing a small software application with all corresponding artefacts like diagrams, program code and documentation, focusing on their relationships and development activities. (2) *Goal-oriented and structured learning* is used to define learning outcomes for every group meeting as a common ground, while *audience response systems* are utilized to motivate the attendance and allow students to self-reflect on their knowledge and competence level. (3) We *harmonize the role of tutors* by holding dedicated teaching workshops for tutors' responsibilities in the programming lab.

The different approaches are evaluated based on surveys for students and tutors over three iterations of the programming lab at TU Dortmund University. Both sides' positive responses and feedback resulted in an enumeration of lessons learned as recommendations and support for other similar courses.

KEYWORDS

learning outcomes, audience response system, worked-out examples, tutoring, programming lab, software engineering

SEENG'22, May 17, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9336-2/22/05.
https://doi.org/10.1145/3528231.3528358

ACM Reference Format:
Malte Mauritz, Stefan Naujokat, Christian Riest, and Till Schallau. 2022. Aligning the Learning Experience in a Project-Based Course: Lessons Learned from the Redesign of a Programming Lab. In *4th International Workshop on Software Engineering Education for the Next Generation (SEENG'22), May 17, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3528231.3528358

1 INTRODUCTION

Creating and developing software applications for the modern world becomes more and more challenging because of the increasing complexity of applications and growing ecosystems of software libraries. It is thus not sufficient to teach future software engineers how to program software. The full software development life cycle comprises applying analysis and design methods, making architecture decisions, as well as testing and deploying the software. Similarly vital for a successful software development process are factors like project planning and working in a team. Our undergraduate Computer Science curriculum at TU Dortmund University schedules the *programming lab* as a mandatory course to facilitate a project-based education for Software Engineering (cf. [8, 15]). Our students work in groups of up to nine people on two software projects that each span the full development life cycle. There are usually ten groups in parallel working on the same assignments, with each group being supervised by one tutor.

Following our goal of a genuine Software Engineering education [5], the programming lab aims to teach students how to apply the software engineering and programming knowledge gained in previous courses in a practical and realistic environment and to understand the 'bigger picture' of the software development process. In addition, they learn the importance of planning and work structure for successfully finishing a group project.

A major challenge for us, the organizers of the programming lab, is to achieve a similar learning experience for all students and to ensure the same learning outcomes in every iteration. However, several influences impact the learning experience in the programming lab.

Student participants come from different subjects with different skill levels, experience, and prior knowledge in the field of software development. Some participants have only rudimentary experience

from their studies, while others already have worked in a professional environment or created software applications in their free time.

Furthermore, each student has its own personality resulting in different levels of involvement and commitment in the group discussion. Some students are very engaged and dedicated, while others are more reserved and diffident.

Differences in experience and personalities of participants directly impact the workflow and dynamics of the student groups in the programming lab. Based on the composition of the individual students, workflows and dynamics differ between groups and also between iterations. For example, experienced students work more independently and tend to create more complex solutions. As a result, less experienced students in the group with problems in understanding the basic design and modelling cannot contribute equally to the development of the application.

The dynamics of groups are further influenced by the tutors' involvement. The management of tutors in the groups' work differ depending on the levels of knowledge about didactic methods and teaching experience, in particular, as they range from undergraduate student assistants to post-docs. While some tutors play the role of an observer and only act when asked for help, others are much more involved with their group's work, up to the point that they behave like an additional group member.

We took over the responsibility of the programming lab in the beginning of the COVID-19 pandemic. Prior programming lab iterations did not provide a uniform learning experience for all students. Even though a schedule with the content of each meeting was provided for the whole course, the team meetings themselves were subject to the personal planning and execution of the tutors, who were, however, not specifically trained for their role in project-based courses. They were only provided by the university with a rudimentary didactic training on teaching of exercise groups.

Furthermore, students were provided with an example implementation to compare and adopt for their development in the programming lab. This example implementation, however, was a too complex project, only consisted of the final product, and did not provide sufficient documentation of the development process and its requirements and design artefacts, i.e., diagrams.

To overcome the said challenges and thus to achieve an alignment of the learning experience for all students of the programming lab, we implemented three didactic improvements based on well-studied concepts. We provide a *worked-out example* of a small software application called *StudyPlanner* as a reference point for the whole development process (Section 3). The example includes all artefacts, program code, documents, and tests for self-learning and testing. Furthermore, for *goal-oriented and structured learning*, we introduced prepared slides with quizzes and examples from the StudyPlanner project for the tutors to present in the group meetings, which give them a standard structure and ensure common learning activities and intended learning outcomes (Section 4). Finally, the third improvement focuses on harmonizing the role of tutors by *schooling tutors for their specific roles and responsibilities* in the programming lab (Section 5).

Please note that we faced a difficult balancing act: on the one hand we wanted to validate that our changes indeed improve the learning experience, while on the other hand we needed to implement them into a mandatory course that processes over 300 students a year. We chose to primarily focus on the latter and thus emphasize here that this paper is an *experience report* rather than a proper empirical study with a control group. While we are, based on our measurements, personal assessment and tutors' reports, confident of having achieved a better learning experience for our students and think that the findings are generally applicable to similar courses, we do not have strong statistical evidence.

The remainder of the paper is structured as follows. Section 2 introduces the didactic concept and structure of the programming lab. We describe the ideas and approaches in detail in Sections 3, 4, and 5. Section 6 presents the evaluation of the feedback from the students and the tutors. We formalized several best practice recommendations for similar courses based on the feedback and evaluation results in Section 7. We finish with a conclusion and possible ways to improve the ideas in Section 8.

2 PROGRAMMING LAB

The programming lab is a mandatory course for students in the 3rd to 5th semester of the undergraduate programs *Computer Science*, *Applied Computer Science*, *teacher training with subject Computer Science*, *Data Science*, and students of *other studies with minor Computer Science* at TU Dortmund University. It takes place twice every semester in two variants: as a course with two meetings in a week during the 15-week-long lecture period and as a course over 6 weeks in the lecture-free period with daily meetings. In the prerequisite courses for participation in the programming lab, students learn the basics of data structures, algorithms, Java programming, system modelling using the Unified Modeling Language (UML) [16], software architectures, design patterns, and software testing. In the programming lab, this content is then practically applied in two projects: a management application and a board game. Alongside these technical skills, students learn to make their first practical steps in project management and team organization. Students work with standard Software Engineering tools, i.e., JetBrains IntelliJ[1] , Git[2], and GitLab[3], as well as Matrix[4] and BigBlueButton[5] for digital team communication.

Up to nine students develop the two software applications as a group supervised by a tutor. The group members are randomly assigned (only considering priorities given for different time slots), in contrast to more complex approaches that aim at an equal distribution of experience level, gender, and language skills [6]. The group meetings are complemented by homework-like assignments and the presentation of development results to the other groups. Development activities in both projects are similar and performed by each student group independently.

Learning Outcome

The goal of the programming lab is to provide students with their first practical insights into the conception and implementation of larger software applications in teams together with project and team management. The learning outcome of the programming

[1] https://www.jetbrains.com/idea/
[2] https://git-scm.com
[3] https://gitlab.com/
[4] https://matrix.org
[5] https://bigbluebutton.org

lab can be described according to the concept of the *Constructive Alignment* by John Biggs [2]:

Students will be able to plan, develop, and implement a software project in a team by

- decomposing a software product into its core aspects based on a requirements specification,
- planning and modelling the software architecture graphically,
- implementing the program using an IDE,
- using the code versioning tool Git within a larger group to ensure an efficient division of programming labour in larger software projects,
- working with modern tools for project planning, which enable to divide complex tasks and form subgroups,
- working constructively in a team of up to nine developers, reflecting on their role and collaboration,

so that they can develop suitable and operational software products as teams in professional practice.

Curriculum

The curriculum of the programming lab comprises two software development projects along a well-structured development process spanning the full life cycle from analysis, design, realization, and testing, to delivery/deployment. The first project requires students to build a *management application*, while for the second, the groups need to realize a *board game*. The tasks for the specific projects are given through application specifications and required features. We do not provide sample solutions and evaluate the individual results.

In the first project, the meetings consist of mini-lectures introducing the teaching content followed by time to work on the project moderated by the tutor or, in later meetings, students of the group. This is extended in the second project to the students' complete management, including designing and planning, whereas the tutors function as stakeholders.

Both projects start with the requirements analysis and the modelling of *use case diagrams* based on an application's specification, resp. the rules of the board game. Identified use cases are refined in *activity diagrams*. Based on this analysis, a class diagram of the application's data is created – the *entity model*. All three models form the *requirements model*. They are demonstrated to the other groups in the first presentation after four group meetings. After the adaption of these models to feedback from other groups, the design phase commences with the extension of the entity model into a complete *structure model* of the application: similar to domain-driven design [7], a service layer is added that together with the entity layer forms the *domain model*. This domain model is further extended by a view layer which models all classes of the application's user interface. After the complete structure of the application is modelled, important functionality is further refined in *sequence diagrams*. The *analysis model* – structure model combined with sequence diagrams – is presented to other groups in a second presentation after seven team meetings. In the remaining time of each project, the groups implement their applications in Java, with corresponding component tests, and write the user documentation. Both application projects end with an evaluation where the other groups inspect, test, and rate the final software application. After the first project, we identify strengths and weaknesses in *reflection meetings* with each group to address these in the development of the second project.

The second project differs from the first in the way that we do not specify the content of each group meeting, but only the dates for the presentation of the *requirements model* and the *analysis model*, including corresponding deadlines for their respective development artefacts. The student groups are responsible to plan their development. In addition to the evaluation of the final product at the end of the project, we also host an *AI tournament* where implemented artificial intelligences for the board games compete.

Our intention as organizers of the programming lab is to provide students with a consistent and comprehensive learning experience for future challenges in academic and professional settings. The following sections present our didactic measures for aligning the students' learning experience in the programming lab despite inconsistent teaching of tutors, diverse backgrounds of students, and varying group dynamics.

3 LEARNING BY WORKED-OUT EXAMPLE

A didactic approach for teaching procedures or workflows is learning by worked-out examples, especially in well-structured domains like mathematics or programming, where the process of finding a solution for a problem follows a defined path of steps [17]. The goal is to illustrate the different phases and give a reference point for the students to use as a basis for their projects. In the following, we first briefly introduce the concept and its theoretical background before illustrating the structure of the work example.

Learning from Worked-Out Examples and Cognitive Load Theory

Learning from worked-out examples is a well-known didactic concept that focuses on learning from one or more examples for a task or process without external influences or guidance. The examples show the workflow step by step with, in most cases, additional information for an easier understanding. The advantage of using worked-out examples in contrast to solving problems as a learning method is explained by work by John Sweller that resulted in the *Cognitive Load Theory* [21]. It states that a problem-solving task creates a high load on the limited capacity of the working memory for understanding the problem and the context. This capacity is then not available for processing the solving steps of the problem that should be learned, which results in reduced learning success.

In contrast, the learning by worked-out examples approach relinquishes on the part of solving tasks and only gives the worked-out examples, so the ineffective load is reduced, and the learner can use all available capacity to understand the solving steps of the process [23]. This train of thought motivates using the example also in the slides of the group meetings (cf. Section 4) to spare time and resources of the students to familiarize with a new example every time.

Worked-Out Example: StudyPlanner

We developed a worked-out example for the software development process to provide learning material that fits students' different skill levels, experiences, prior knowledge, and personalities along with different group dynamics and workflows.

The example consists of a complete application with additional diagrams, documentation, tests and explanations. The application

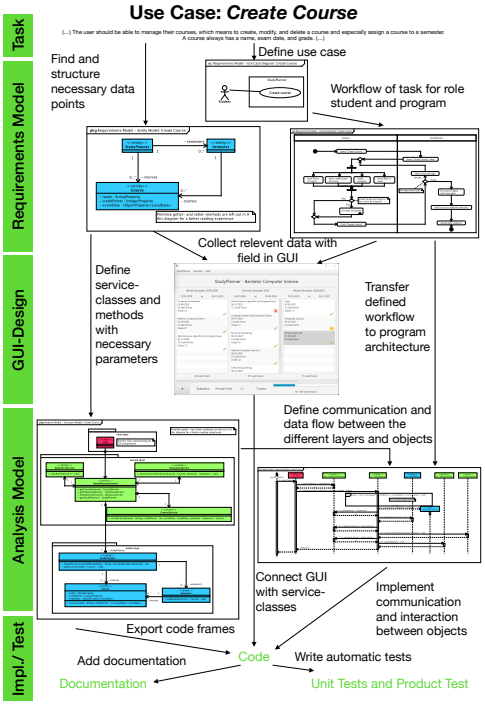


Figure 1: Different phases of the software development process at the use case *Create Course* with focus on the connections and work steps between the artefacts.

with the name *StudyPlanner* is a software tool that supports the user to manage a curriculum with different modules over several semesters. It also provides statistics over the number of collected credit points, the state of a module, the date of an exam, and the current average grade.

We used the worked-out example in three different approaches to give students a wide range of perspectives on the development process, so they can choose the right way for them.

In the first approach, the example is used in slides for different group meetings to illustrate significant results and in quizzes to interact with the students. This attempt is explained in detail in Section 4.

In the second approach, all artefacts for the complete example like diagrams, program code, tests, documentation for program code and tests are available on a website grouped by the different phases. This includes diagrams, documentation, tests, and the program code with additional comments for better understanding. The program itself can be downloaded from a git repository[6] and executed for testing or modification. Additional comments are added to explain design decisions or unclear parts.

In the third approach, only the two individual use-cases *Create Course* [7] and *Show Statistics* [8] are used to illustrate the connections and working steps between the artefacts in the different phases instead of using the whole example. This is shown in Figure 1 for the use-case *Create Course*. The example starts with the section **Task** with a text describing the use case. For the **Requirements Model**, the use case diagram with the use case *Create Course* is defined along with a domain model that contains all necessary data points and relationships to represent a new course in the program. The associated activity diagram further illustrates the workflow for the user to create a new course and actions processed inside the application. In the next phase, the graphical user interface is designed to match the defined workflow and to provide input fields for all data points. In the **Analysis Model** phase, the domain model is extended to represent the entire structure of the program. A sequence diagram shows the communication between the different objects following the program structure based on the predefined workflow from the activity diagram. The class diagram provides the code frame for the implementation. The implementation follows the defined path in the sequence diagram.

4 STRUCTURING OF THE MATERIAL BASED ON LEARNING OUTCOMES

Based on the observations of previous iterations of the programming lab, overhauling the used material to approach the challenges presented in Section 1 was done along two primary goals:

Introduction of structured slides
Each student should achieve the same learning outcomes. Adjust the tutors' material to align the taught knowledge across all groups.

Supporting students' competence assessment
Provide continuous competence assessments. The students are then able to self-reflect on their knowledge and competence level.

These two goals and the methods used to achieve them are explained in more detail in the following paragraphs.

Introduction of structured slides
As stated in Section 2, the programming lab has already defined learning outcomes for the whole course. However, the specific topics and goals of each session were previously not present. This led to broadly different interpretations and execution amongst the tutors which ultimately led to varying project results and learning outcomes for the students.

The introduction of structured slides is used to counteract these problems by defining concrete learning outcomes in the form of *Constructive Alignment* [2] for each session of the programming lab. With this technique, students, and tutors always know which topic and methods should be taught in each session. Additionally, we provide dedicated handouts for the tutors detailing the goals and content. An example for such learning outcomes from an individual session's slides about activity diagrams is the following:

Students will be able to model the process of a use case based on a given task by

- defining the actors involved,
- formulating the necessary actions in an abstract form,
- linking the individual actions logically,

so that they can define the interactions of the user and delimit the responsibility of the user from that of the software.

[6] https://github.com/tudo-aqua/study-planner
[7] https://github.com/tudo-aqua/study-planner#example-1-create-course
[8] https://github.com/tudo-aqua/study-planner#example-2-show-statistics

Each session in the first project starts by introducing new topics with their respective learning outcomes. By using these introductions at the beginning of each session, we are following the theory of the *informative lesson introduction* [11]. By always introducing the sessions' content at the beginning and by adapting the sessions' tasks, the students should have all necessary information available when needed. In the best case, they should also be able to apply the new and theoretical knowledge to practical tasks.

The introduction of learning outcomes at the beginning of each session helps the students to get an overview of upcoming topics and how they fit into the greater Software Engineering context. We introduce a mechanism for self-assessments during the sessions to self-reflect on their understanding of the content.

Supporting students' competence assessment and development

To enable self-assessment for the students the provided slides include small quizzes about the content introduced in each session. They are located after each content block and contain questions about the newly learned methods and paradigms. The quizzes vary from finding semantic issues in given diagrams, to deciding which software construct should be part of the final program and fixing syntactical errors. They are always based on the continuous StudyPlanner example introduced in Section 3. A quiz would ask the students to find errors in a given diagram, based on the definition they learned in the related session.

These quizzes can normally be answered verbally, but the ongoing COVID-19 pandemic led to the introduction of *audience response systems (ARS)* [3, 12]. These can be used to get direct feedback from the students through the usage of handheld devices on which the asked question is displayed. The collected answers are shown to the tutor who then can react, correct and give further explanations. We implemented the quizzes through Mentimeter[9], quick surveys as multiple-choice questions, and the shared whiteboard feature of the used meeting software. With these methods, the students can reflect on their knowledge and assess whether they understood everything correctly. Another positive side effect of ARS is the activation of inactive students [10], especially during the pandemic's remote sessions.

5 HARMONIZING THE ROLE OF TUTORS

The impact from alignments of meetings to learning outcomes and unification of examples in the programming lab to the learning experience of students is influenced by the teaching quality of the tutors. As described in Section 1, we organizers recognized a varying quality in the teaching among tutors with a negative impact on the learning experience of students in the past. To ensure an equal learning experience among all students, we identified the need to harmonize the roles and responsibilities of tutors in the programming lab.

We first specified the roles of tutors and their responsibilities within the programming lab. We identified six roles and their main tasks for tutors in the programming lab. Tutors have to (1) support students in their learning (coach), (2) moderate group discussion (moderator), (3) grade students' results (auditor), and (4) provide assistance for technical and implementation problems (expert). They (5) oversee the project progress of their groups (project manager) and (6) steer the work of groups to result in an operable software product (product manager).

Based on these six roles, eight essential responsibilities and corresponding skills have been identified for tutors in the programming lab: (1) learning outcomes, (2) group management, (3) motivation, (4) conflict management, (5) minimal assistance, (6) moderation of discussions, (7) grading of student performance, and (8) internal communication.

We held four workshops on the topics (1) self-reflection of general teaching, (2) minimal assistance, (3) motivation and activation, and (4) conflict management to teach tutors the corresponding skill sets. All workshops lasted about 60 minutes and were held online. Contributions by tutors were organized and visualized using the online-tool Padlet[10].

The structures of all workshops were similar and inspired by the didactic concepts *learning by doing* [20] and *learning by example* [22]. We planned each workshop intending to stimulate the exchange of experience between tutors and maximize the later application of presented didactic methods by tutors in their teaching. Each workshop started with a self-reflection unit about personal experiences. Afterwards, we presented theoretical concepts to the individual topics, which the tutors applied together to their experiences.

The following paragraphs describe each workshop in detail.

Workshop 1: Self-Reflection

The workshop series started with a unit on biographic reflection of personal teaching experiences (cf. [13]). This workshop was envisaged to provide an overview of the responsibilities of tutors in the programming lab and a basis for upcoming workshops.

The tutors were held to gather experiences from their work in the programming lab or other courses, discuss their experiences, and categorize them based on the nine responsibilities of tutors in the programming lab. Eight tutors contributed 22 personal experiences throughout all nine responsibilities.

Workshop 2: Minimal Assistance

The second workshop addresses the topic of minimal assistance [1, 24]. Learning outcomes of students are subject to the correct intervention of tutors. In the case of student problems, tutors must intervene and provide students with minimal information for their assignments that these students are able to continue to solve the assignment on their own. In addition to information scope, also the time of intervention is important. A late intervention may lead to frustration for struggling students and, therefore, decrease their motivation and participation in the programming lab.

In the workshop, tutors were asked to list personal experiences from their teaching, when and how they helped students. These experiences were then categorized into the five levels of staged intervention by Zech (cf. [24]): (1) motivational level, (2) feedback level, (3) general strategic assistance, (4) associative strategic assistance, and (5) associative assistance. As a result of the workshop, a plan for the identified teaching situations with corresponding assistance actions has been established, which the tutors can consider for assisting students within their teaching.

[9] https://mentimeter.com

[10] https://www.padlet.com

Workshop 3: Motivation and Activation
The third workshop addresses the motivation and activation of students to participate equally in the group discussion as well as in the implementation of the software program. Active participation of students in the programming lab is a prerequisite for successful learning outcomes of students. However, not all students are equally involved due to, e.g., their reserved personalities or limited experience in Software Engineering.

In the workshop, tutors were asked to reflect upon their actions to provide a safe environment and to motivate their students to actively participate. Identified methods were categorized to the basic orientations of human behaviours in the *Riemann-Thomann-Model* [18]: (1) distance alignment, (2) proximity orientation, (3) permanent alignment, and (4) alternating orientation. Each of these categories imposes different requirements for teaching by tutors in the programming lab. While humans of category *distance alignment* prefer independence and individuality of independent tests and homework, humans with the need for proximity desire the social communication and interaction of group work. Permanent aligned humans strive for control, persistence, and organization through thoroughly planning and repeated activities, opposed to humans with an alternating orientation who favour the variety, spontaneity, and creativity from changing assignments and interactions.

The workshop further addressed the problem of social loafing (cf. [14]) and cultural distinctions of allocentric and idiocentric cultures regarding active participation (cf. [4]).

Workshop 4: Conflict Management
The fourth workshop aimed to address common conflicts within the programming lab and to provide tutors with strategies to address and mitigate these conflicts. In the past, tutors failed to sufficiently address emerging conflicts in their groups due to missing knowledge of appropriate conflict solution strategies. However, it is important that tutors address and solve conflicts timely to prevent any manifestation of conflict within the groups and avoid the reduction of perceived safety, participation, and learning outcomes of students (cf. [9]).

The workshop started with a distinction of (1) social conflicts, (2) internal conflicts, and (3) structural conflicts (cf. [9]). Only social conflicts are of interest for the programming lab, as we focus on the conflict between students within a group and conflicts between students and tutors, with the target to ensure productive group work throughout the programming lab.

We then asked the tutors to reflect on their experiences with social conflicts. Nine conflicts were gathered and categorized into the interpersonal conflict types: (1) goal conflict, (2) distributional conflict, (3) relationship conflict, (4) role conflict, and (5) perception-oriented conflict (cf. [19]). Following the conflict escalation by Gasl [9], *win-win* solutions were identified for each conflict. Finally, we organizers urged the tutors to remain objective and refrain from personal interpretations throughout conflicts to make themselves not attackable by either conflicting party.

6 EVALUATION

We evaluated our improvements to the programming lab throughout three iterations in 2021. To measure the impact of the worked-out example and slides (cf. Sections 3 and 4), we split the groups into two sets: in the first, the groups used the example, slides, and infrastructure of quizzes and ARS, whereas the second set of groups was taught normally without these changes. To reduce the impact of the tutors' backgrounds, we distributed their groups evenly according to their experience in teaching the programming lab and furthermore ensured that no tutor had a group with the original approach followed by one using the improved methods and material. As already after two iterations we were very confident that our changes indeed improved the learning experience, we did not apply the splitting again to the third iteration to not deliberately disadvantage half of the students.

For the tutor schooling, we introduced all tutors after the first iteration to the description of the tutor role and educated them in the four workshops (cf. Section 5).

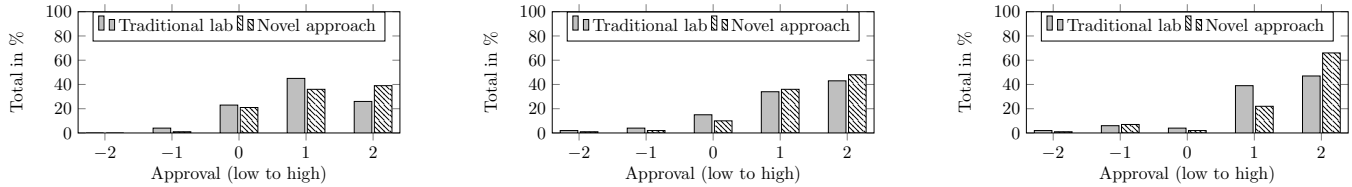
We created two surveys with 28 questions for students and 32 questions for tutors to gather feedback from students about their learning experience and tutors about their teaching experience. The feedback from these surveys provided us with insights about the helpfulness of the applied teaching methods for the development and problem-solving of students from the viewpoints of students and tutors.

Over the course of three iterations, 164 students and 25 tutors in total have answered our surveys. Students as well as tutors already valued the programming lab without our improvements. However, the feedback from groups which used the presented didactic changes has been even more positive. The results indicate that our improvements allow the programming lab to be held more effectively for both students and tutors to create a better learning experience. We have selected six of the most relevant questions from the total 60 survey questions for demonstrating the impact of our approaches.

Students: Clear definition of learning outcomes
Firstly, Figure 2a shows the benefit of using structured slides with learning outcomes for each session (cf. Section 4). The results indicate that by introducing structured slides we were able to improve the clearness of the individual learning outcomes of each session in groups using the slides.

Students: Introduction of a continuous example
The results (cf. Figure 2b) of groups using slides with predefined examples, as described in Section 3, show not only that the examples were received better when predefined. They also demonstrate that examples in general are a useful method to pass on knowledge. The small increase between the two groups stems from the common usage of examples by the tutors. In summary, around 81% of the students agree to the fact that using examples is improving their understanding of the topic. This underlines the importance of examples for such classes as explained in Section 3.

Students: Management ability of tutors
One of the desired outcomes of our approach is the alignment of teaching behaviours and applied methods for the tutors. For that purpose, besides providing slides and specific infrastructure, we also tried to harmonize their role, as described in Section 5. The combination of both methods leads to an increase in the tutors' rating regarding their management of the sessions (cf. Figure 2c).



(a) Question: Were you always aware of the sessions' intended learning outcome?

(b) Question: How did the examples help you to understand the sessions' content?

(c) Question: How do you rate the management of the sessions by your tutor?

Figure 2: Survey results from student participants. Higher values mean a higher agreement to the asked question.

It is important to consider that the personality and personal administration of each tutor influences this feedback. However, we are confident that the tendency is in favour of our approach.

Based on the harmonization of the tutor's role in the programming lab the second part of our survey focussed on the perception of the tutors. Again, the tutors were also split into the two groups already discussed above.

Tutors: Availability of information
According to Figure 3a there is a positive trend in the availability of necessary information. Over 75% of tutors using slides reported having all necessary information for each session. In contrast, only 50% of tutors without slides stated the same. This shows that by providing examples, quizzes, learning outcomes and general information material (cf. Section 3 and 4) we were able to improve finding the necessary information.

Tutors: Clearly defined tasks per session
Figure 3b shows that 69% of the tutors using slides are aware of their tasks in each group session, whereas only 50% of tutors without slides state to know their tasks. Both groups contained inexperienced tutors. The results demonstrate that using slides with related handouts improved the awareness of the predefined tasks for each session. As a result, we see the effect of harmonizing the tutors throughout the whole programming lab.

Tutors: Teaching the intended content
While overall the tutors are quite confident in teaching the intended content, Figure 3c indicates that tutors using slides had fewer problems than the tutors without slides. This improvement originates from providing structured material and instructions given to the tutors prior to the course.

7 LESSONS LEARNED

This experience report with its promising results could help other teachers, organizers and tutors of similar labs by adapting our findings to their courses. For this reason, we are summarizing our lessons learned from our didactic overhaul of the programming lab.

Use a continuous example
The use of a continuous example throughout the whole lab helped the students to follow our thought development process. It is also easier to understand new additions and concepts applied to the same example instead of having to rethink the example every time.

Provide structured material
Providing structured material for the tutors aligns the pacing and description of topics. Additionally, new tutors can take the material to prepare sessions properly and have an easier time getting started.

Define learning outcomes of each session
For each session, the learning outcomes should be available and presented to the students and tutors. This helps to set the focus of the session onto specific parts of the content. It also declares which information can just be seen as knowledge and which should be applied later on.

Include tutors in the process
The tutors should be trained to correctly transfer information and should always know their role in the course. Additionally, they should be brought into the process of improving the course, as they are the ones teaching and should, therefore, bring in their own experience.

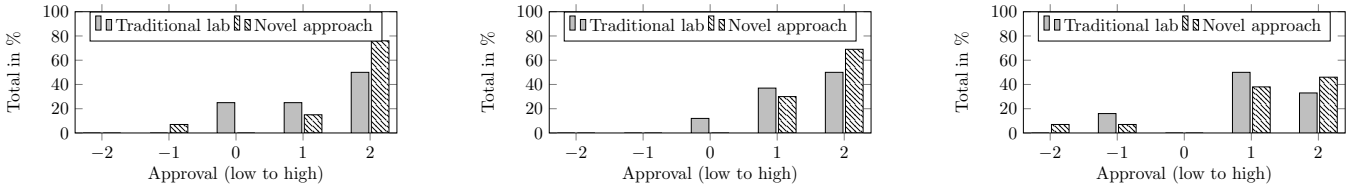
8 CONCLUSION

The goal to provide all students with an equivalent positive learning experience in our project-based programming lab has been challenging over the last years by different levels of knowledge and skill among students, unequal compositions of teams, and diverse approaches to teaching among tutors.

We have introduced three approaches in the programming lab based on well-known didactic concepts to address these challenges: First, we introduced a common example of a study planner project for the complete development process with all UML diagrams and implementation in Java. Second, we aligned the meetings among all groups by introducing prepared slides with the intended learning outcomes as well as quizzes and surveys. Finally, we harmonized the role of tutors in the programming lab by teaching tutors their responsibilities, i.e., in general teaching, minimal assistance, activation and motivation, and conflict management, in four workshops.

Throughout three iterations of the programming lab in 2021 we gathered the feedback of 164 students and 25 tutors by surveys. Students honoured the improved communication of each meeting's intended learning outcome, the examples used in the meetings to introduce individual development activities, and the better session lead of tutors. Tutors valued the improved teaching of students by the provided examples and the predefined content of each meeting. Based on this positive feedback, the improvements (with some extensions and refinements) are now permanently included into the course and have led to increased ratings in the teaching evaluation the faculty conducts each semester.

In the most recent iteration, we introduced further improvements to the programming lab. We replaced Java with Kotlin, which is



(a) Question: How easily were you able to find the information for each session?

(b) Question: Were the tasks for each session always clearly defined and understandable?

(c) Question: How comfortable did you feel teaching your students the intended content of each group meeting?

Figure 3: Survey results from the lab's tutors. Higher values mean a higher agreement to the asked question.

becoming more and more popular in education[11]. It proves to be a modern programming language that is easy to learn and teach, but remains in the Java ecosystem already known to our students. Furthermore, the effort for developing the GUI, which so far was realized with JavaFX, was disproportionately high with only little additional learning outcomes. We instead developed and used BoardGameWork[12], a lightweight Kotlin UI framework for board games. This streamlining allowed us to add further content to the lab, such as teaching a Git branching model or advanced IDE features, e.g., refactoring or debugging. The informal feedback from students and tutors so far has been overwhelmingly positive.

Until today, the didactic improvements have only been applied in an online format of the programming lab due to the COVID-19 pandemic. However, we are confident that these didactic changes provide a similar impact on the learning experience in the usual in-person environment of the programming lab.

AUTHORS' BIOGRAPHIES

All authors research and teach (as post-docs and PhD students) at the chair for Software Engineering of the Computer Science department at TU Dortmund University, where we recently took responsibility for organizing the programming lab. To modernize and streamline the lab, we applied our knowledge from previous teaching experience in Software Engineering courses and a didactic certificate program. Our primary goal is to improve the lab to effectively align the students' learning outcome among all groups under heterogeneous conditions by continuously modernizing the programming lab and applying further didactic improvements.

REFERENCES

[1] Hans Aebli. 1965. Grundformen des Lehrens: ein Beitrag zur psychologischen Grundlegung der Unterrichtsmethode. (1965).
[2] John Biggs. 1996. Enhancing Teaching Through Constructive Alignment. *Higher Education* 32 (10 1996), 347–364. https://doi.org/10.1007/BF00138871
[3] Jane E Caldwell. 2007. Clickers in the large classroom: Current research and best-practice tips. *CBE—Life Sciences Education* 6, 1 (2007), 9–20. https://doi.org/10.1187/cbe.06-12-0205
[4] Catherine L Caldwell-Harris and Ayse Aycicegi. 2006. When personality and culture clash: The psychological distress of allocentrics in an individualist culture and idiocentrics in a collectivist culture. *Transcultural psychiatry* 43, 3 (2006), 331–361.
[5] Simon Dierl, Falk Howar, Malte Mues, Stefan Naujokat, and Till Schallau. 2021. Do Away with the Frankensteinian Programs! A Proposal for a Genuine SE Education. In *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*. 26–30. https://doi.org/10.1109/SEENG53126.2021.00012
[6] Dora Dzvonyar, Lukas Alperowitz, Dominic Henze, and Bernd Bruegge. 2018. Team Composition in Software Engineering Project Courses. In *2018 IEEE/ACM International Workshop on Software Engineering Education for Millennials (SEEM)*. 16–23.
[7] Eric Evans. 2003. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[8] Maria Lydia Fioravanti, Bruno Sena, Leo Natan Paschoal, Laíza R. Silva, Ana P. Allian, Elisa Y. Nakagawa, Simone R.S. Souza, Seiji Isotani, and Ellen F. Barbosa. 2018. Integrating Project Based Learning and Project Management for Software Engineering Teaching: An Experience Report. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 806–811. https://doi.org/10.1145/3159450.3159599
[9] Friedrich Glasl. [n. d.]. *Konfliktmanagement ein Handbuch für Führungskräfte, Beraterinnen und Berater*. HauptVerl. Freies Geistesleben.
[10] Charles R. Graham, Tonya R. Tripp, Larry Seawright, and Georgel. Joeckel. 2007. Empowering or compelling reluctant participators using audience response systems. *Active Learning in Higher Education* 8, 3 (2007), 233–258. https://doi.org/10.1177/1469787407081885 arXiv:https://doi.org/10.1177/1469787407081885
[11] Jochen Grell and Monika Grell. 1996. *Unterrichtsrezepte*. Beltz. https://ixtheo.de/Record/216344956, zuletzt geprüft am 25.08.2021.
[12] Robin H. Kay and Ann LeSage. 2009. Examining the benefits and challenges of using audience response systems: A review of the literature. *Computers & Education* 53, 3 (2009), 819–827. https://doi.org/10.1016/j.compedu.2009.05.001
[13] M.V. Klementyeva. 2016. Biographical Self-Reflection as a Resource for Personal Development in Adults. *Cultural-Historical Psychology* 12 (04 2016), 14–23. https://doi.org/10.17759/chp.2016120102
[14] Bibb Latané, Kipling Williams, and Stephen Harkins. [n. d.]. Many hands make light the work: The causes and consequences of social loafing. 37, 6 ([n. d.]), 822.
[15] Paula Morais, María João Ferreira, and Bruno Veloso. 2021. Improving Student Engagement With Project-Based Learning: A Case Study in Software Engineering. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje* 16, 1 (2021), 21–28. https://doi.org/10.1109/RITA.2021.3052677
[16] Bernd Oestereich and Axel Scheithauer. 2013. *Analyse und Design mit der UML 2.5*. Oldenbourg Wissenschaftsverlag.
[17] A Renkl. 2002. Worked-out examples: instructional explanations support learning by self-explanations. *Learning and Instruction* 12, 5 (2002), 529–556. https://doi.org/10.1016/S0959-4752(01)00030-5
[18] Fritz Riemann. 2011. *Grundformen der Angst*. E. Reinhardt.
[19] Siegfried Rosner. [n. d.]. *Gelingende Kommunikation - revisited: Ein Leitfaden für partnerorientierte Gesprächsführung, wertschöpfende Verhandlungsführung und lösungsfokussierte Konfliktbearbeitung*. Rainer Hampp Verlag.
[20] Roger C Schank, Tamara R Berman, and Kimberli A Macpherson. [n. d.]. Learning by doing. 2, 2 ([n. d.]), 161–181.
[21] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.
[22] J Gregory Trafton and Brian J Reiser. 1993. Studying examples and solving problems: Contributions to skill acquisition. In *Proceedings of the 15th conference of the Cognitive Science Society*. Citeseer, 1017–1022.
[23] Tamara Van Gog, Fred Paas, and John Sweller. 2010. Cognitive load theory: Advances in research on worked examples, animations, and cognitive load measurement. *Educational Psychology Review* 22, 4 (2010), 375–378.
[24] Friedrich Zech. [n. d.]. *Grundkurs Mathematikdidaktik: theoretische und praktische Anleitungen für das Lehren und Lernen im Fach Mathematik*. Beltz.

[11] https://kotlinlang.org/education
[12] https://github.com/tudo-aqua/bgw