Enhancing Software Engineering Education Through Experimentation: An Experience Report

Marco Kuhrmann
Clausthal University of Technology
Institute for Applied Software Systems Engineering
Goslar, Germany
kuhrmann@acm.org

Jürgen Münch
Reutlingen University
Business Informatics & Herman Hollerith Center
Böblingen, Germany
j.muench@computer.org

***Abstract*—Software engineering courses have to deliver theoretical and technical knowledge and skills while establishing links to practice. However, due to course goals or resource limitations, it is not always possible or even meaningful to set up complete projects and let students work on a real piece of software. For instance, if students shall understand the impact of group dynamics on productivity, a particular software to be developed is of less interest than an environment in which students can learn about team-related phenomena. To address this issue, we use experimentation as a teaching tool in software engineering courses. Experiments help to precisely characterize and study a problem in a systematic way, to observe phenomena, and to develop and evaluate solutions. Furthermore, experiments help establishing short feedback and learning cycles, and they also allow for experiencing risk and failure scenarios in a controlled environment. In this paper, we report on three courses in which we implemented different experiments and we share our experiences and lessons learned. Using these courses, we demonstrate how to use classroom experiments, and we provide a discussion on the feasibility based on formal and informal course evaluations. This experience report thus aims to help teachers integrating small- and medium-sized experiments in their courses.**

***Index Terms*—software engineering education; empirical software engineering; experimentation; empirical studies; experience report**

I. Introduction

Software engineering aims at systematically developing software-intensive systems and, as a discipline, comprises the two fields *software technology* (analysis, design, coding, etc.) and *software management* (software process, project management, effort estimation, etc.). Software engineering is considered an interdisciplinary field thus exposing engineers to many needs from different domains, which all rely on software. This makes software engineering complex and, thus, makes it a subject hard to teach. Even though several guidelines such as [1]–[4] describe curricula covering software engineering in its entirety, still, many software engineering curricula are focused on the core topics from computer science, i.e., the software technology parts, but provide a rudimentary software engineering toolbox only, e.g., one bigger semester project or a project management course. Moreover, quite often, students lack the opportunity to experience real-wold problems (e.g., systems of considerable size or large-scale projects with moaning customers) or to work in a safe environment that supports learning from failure (e.g., work under heavy stress, problem-handling in teams, or project failure). Humphrey et al. [5] state: "*with few exceptions, the reasons that large-scale development programs have failed have not been technical [...]. As the cancellation of two large and critical efforts demonstrates, these systems have almost always failed because of program-management problems*." That is, students are often left unprepared for todays heterogenous and often distributed/virtual project environments.

But how to teach software engineering in the current, yet still changing environment to better prepare the students? A promising route is to run projects with real clients from industry as described by Brügge et al. [6]. This provides students with deep insights into technology, product development, and team work. However, as we already discussed in [7], *product-focused projects* are focused on *delivering a product* on time. That is, the actual project—and all associated methodical questions—are likely to be shifted into the background as the focus is on code and pleasing the client. At the same time, team structure [8], people's personality [9], [10], or general soft factors [11] affect a team's performance significantly and, thus, also impact the quality of the product under development [12]. That is, ironically, projects have to be considered critical to teach software engineering beyond product development and delivery. Nonetheless, projects are a suitable instrument to help students experience themselves the effects of the software engineering principles or techniques and methods, and to better understand their practical relevance.

Experiments are a tool allowing for teaching specific aspects in a controlled and straightforward manner. Guidelines for experimentation in software engineering are available [13], yet, experiments are nowadays mainly used to conduct research, but they are barely used as teaching tools [14].

Problem Statement and Objective: In this paper, we address the problem of finding adequate means for teaching software engineering in an efficient and effective manner. Efficiency must be reached by tasks that allow for experiencing a phenomenon, and analyzing and understanding it quickly (i.e., what happens and why?). Effectiveness must be reached by making knowledge obtained sustainable (i.e., detecting patterns in problems and transfer/apply knowledge). We aim at providing students with an environment in which they can

learn *and* experience phenomena by using experiments, and we aim to provide an instrument that helps teachers integrating small- and medium-sized experiments into their courses to allow for collecting empirical data that students can use to link evidence with their personal experience. We also aim to train students to use systematic ways of work that help grounding decisions in evidence to, for instance, analyzing and improving failure scenarios [15] or to steer continuous quality improvement programs.

Contribution: Grounded in more than five years of experience in using experimentation as a teaching tool, in this paper, we report our experience and lessons learned. The paper at hand wraps up our concept proposal published in 2012 [16], by collecting and aggregating experiences made so far. Specifically, we review our previously published material [7], [17]–[20], and we add recent experiences. We provide examples and we condense our lessons learned into tipps and guidelines that help teachers improving the use of empirical studies in their own courses.

Outline: The remainder of this paper is organized as follows: Section II summarizes related work. The approach for experimentation-based teaching and an overview of the courses in which we implemented this approach are shown in Section III. Section IV presents the evaluation of the approach and discusses our experiences and lessons learned. We conclude the paper in Section V.

II. Related Work

The idea to use experimentation (or more general: empirical studies) as a means to teach software engineering is not new: Basili et al. [21] were among the first presenting a framework and a process for experimentation for software engineering. Experimentation was, however, mainly used for research purposes (with all challenges and risks as discussed by Runeson [22]), but got only little appreciation as a teaching tool. Using experiments in teaching usually focuses on conducting experiments as part of a regular course with students acting as experimental subjects. Typical objectives of such experiments consider comparisons of different quality assurance processes [23], Global Software Engineering [24]–[26], or software engineering in general, e.g., [27]–[30]. The experimental treatments selected are usually engineering level processes, such as development techniques [31], modeling techniques, or test approaches, e.g., [32]–[34].

Over the years, experimentation as a means to improve teaching has proved successful in many disciplines[1]. For instance, Parker [36] mentions experiments have become widespread teaching tools in economics in the 1990s. Nowadays, many economists use experiments as educational tools and mention several benefits. In particular, experiments are distinctive, participative, and students are likely to remember lessons associated with experiments and consider experiments fun. Dillon [37] draws an important conclusion from his overview of advantages and disadvantages of experiments: successful observation of a phenomenon as part of an empirical study should not be an end in itself. Instead, students should have enough time to get familiar with the ideas and concepts associated with the phenomenon of interest. This also leads to a discussion on the suitability of experiments in teaching [14], [38]. For instance, to make experiments a useful tool, Parker [36] recommends (i) aligning experiments with the central course topic, (ii) selecting a concept that should not be easily understood without the experiment or already obvious, and (iii) enable students to quickly learn the necessary prerequisites for participating in the experiment.

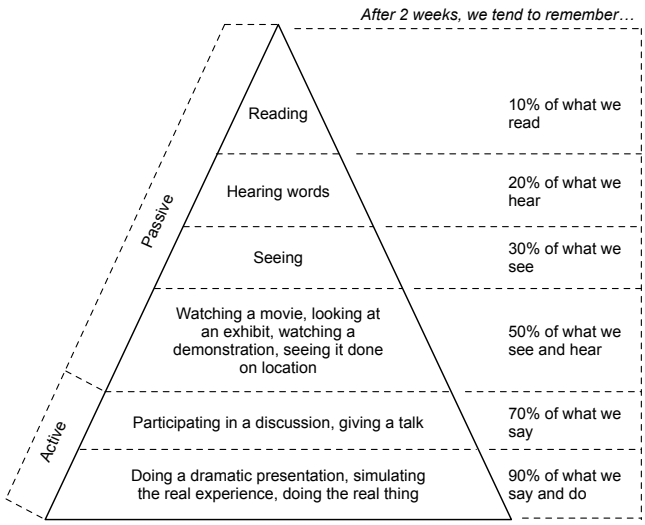

Fig. 1. Dale's cone of learning (according to [39]).

Referring to Dale's cone of learning (see Fig. 1; [39]), experiments—also projects—are located "at the bottom" of the cone which comprises the three elements *doing a dramatic presentation*, *simulating the real experience*, and *doing the real thing* in the category "Doing". In line with Parker [36], Dale claims that students remember about 90% of the contents due to hands-on training. Therefore, software engineering education usually comprises practical work. For example, Gnatz et al. present an approach to combine theory and hands-on lab sessions; a pattern that is also applied in [25], [27]–[29]. Those courses mainly address the topics of project management and software development in teams, external or internal clients in one- or multi-site settings [24], [40]. External clients are favored by Brügge et al. [6], who describe large-scale courses in which student teams run through a full development cycle working on project ideas sponsored by industrial clients.

Closely related are educational games [41], which are adapted to exercise Lean and agile software development practices. These approaches also integrate continuous experimentation and close collaboration between academia and

[1] An important source for classroom experiments is the 'SERC Portal for Pedagogy in Action', created by Ball et al. [35]. The repository includes a comprehensive list of experiments from different disciplines that can be used for replication in classroom settings. In addition, it contains references to scientific studies that provide empirical evidence about the expected positive effects of experiments as teaching tools.

TABLE I
OVERVIEW OF THE EXAMPLE COURSES IMPLEMENTING THE EXPERIMENTATION-BASED TEACHING APPROACH.

Course	Id	Description
Software Engineering Processes	SEP	The goal of the course "Software Engineering Processes" (SEP, Fig. 3-1) is to teach students how to deal with the construction and the continuous improvement of software processes. The course addresses the whole software process lifecycle and the (common) tasks needed to be performed for analyzing, conceptualizing, designing, implementing, publishing, and assessing a software process. We implemented a *semester-spanning case study* in this course. This course was implemented at the Master's level, and the group size was limited to 15 students. We refer to [17], [18] as previously published material.
Agile Project Management and Software Development	APM	The goal of the course "Agile Project Management and Software Development" (APM, Fig. 3-2) is to teach students advanced project management techniques with a special focus on agile project management and group dynamics. The course covers all major project management and development fields (e.g., requirements engineering, architecture and design, development and quality assurance techniques, and globally distributed software development). We implemented two *controlled experiments* in this course. This course was implemented at the Master's level, and the group size was limited to 24 students. We refer to [7], [19], [20] as previously published material.
Software Quality Management	SQM	The goal of the course "Software Quality Management" (SQM, Fig. 3-3) is to teach students advanced quality management techniques with a special focus on software test, code quality, and organization-wide quality management (standards). The course covers selected fields in software testing and quality management (e.g., quality models, standards, software testing techniques, quality management tools). We implemented one *controlled experiment* in this course. This course was implemented at the Master's level, and the group size was limited to 15 students.

industry with the principles of Lean software and product development, e.g., [42], [43]. Also, a simulation can help playing "what if games" and, thus, helps better understanding decision-making or management processes [44]. For example, Mandl-Strieglitz [45] proposes a simulation-based approach to teach students project management providing them with the opportunity to make "real" experiences, similar to what we have reported in the context of agile project management and distributed software development [46] and in [7], [20].

The experiences presented in this paper differ from the "normal" implementation of experiments. Conducting research is possible as we demonstrated in our case studies [17], [47], yet, our focus of using experimentation is to make students learn from their own experiences and to associate the experiences with data that they have collected themselves, rather than convincing them to 'just' trust in external findings (referring to [39]: "Verbal Receiving" versus "Doing").

III. EXPERIMENTATION-BASED TEACHING IN PRACTICE

We introduce our approach for experimentation-based teaching as an experience-based generalization. We provide an overview of the approach in Section III-A. Section III-B provides an overview of the courses we organized following this approach.

A. General Organization

Figure 2 provides an overview of the overall course organization. In general, we organize our courses in three phases: In the first three weeks, teachers set scene the by introducing the course and providing the fundamentals and basic knowledge required, usually in a classic classroom setting. In the second phase, i.e., weeks 4 to *n*-3, teachers provide theory and context,

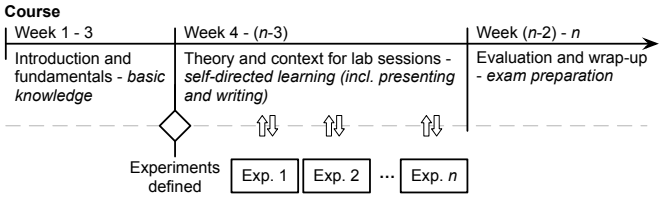

Fig. 2. Structure for integrating experiments in the teaching environment.

and students work on selected topics in a self-directed manner. This work can be performed individually or in small groups. Students have to prepare small presentations, which the teacher needs to include in the overall course schedule, and they have to write short essays—mainly for sharing their findings with the other students. The topic selection and assignment needs to be done at the beginning of the second phase (milestone: *experiments defined*). The second building block in this phase is the experimentation part, which needs to be aligned with the respective course topics. A number of assignments defines the experiments and, therefore, teachers must ensure that the students have obtained all pre-knowledge required to work on the assignments, and teachers have to organize the sessions and have to align the tasks with the individual sessions. For instance, if a classroom is available for a 4-hour block, teachers have to tailor experiments such that they fit into this timeframe. For more comprehensive tasks such as consecutive case studies, teachers have to design experiments with pre-defined breaking points, which are aligned with the respectively required theoretical background. The third phase (weeks *n*-2 to *n*) concludes the course. That is, the course is

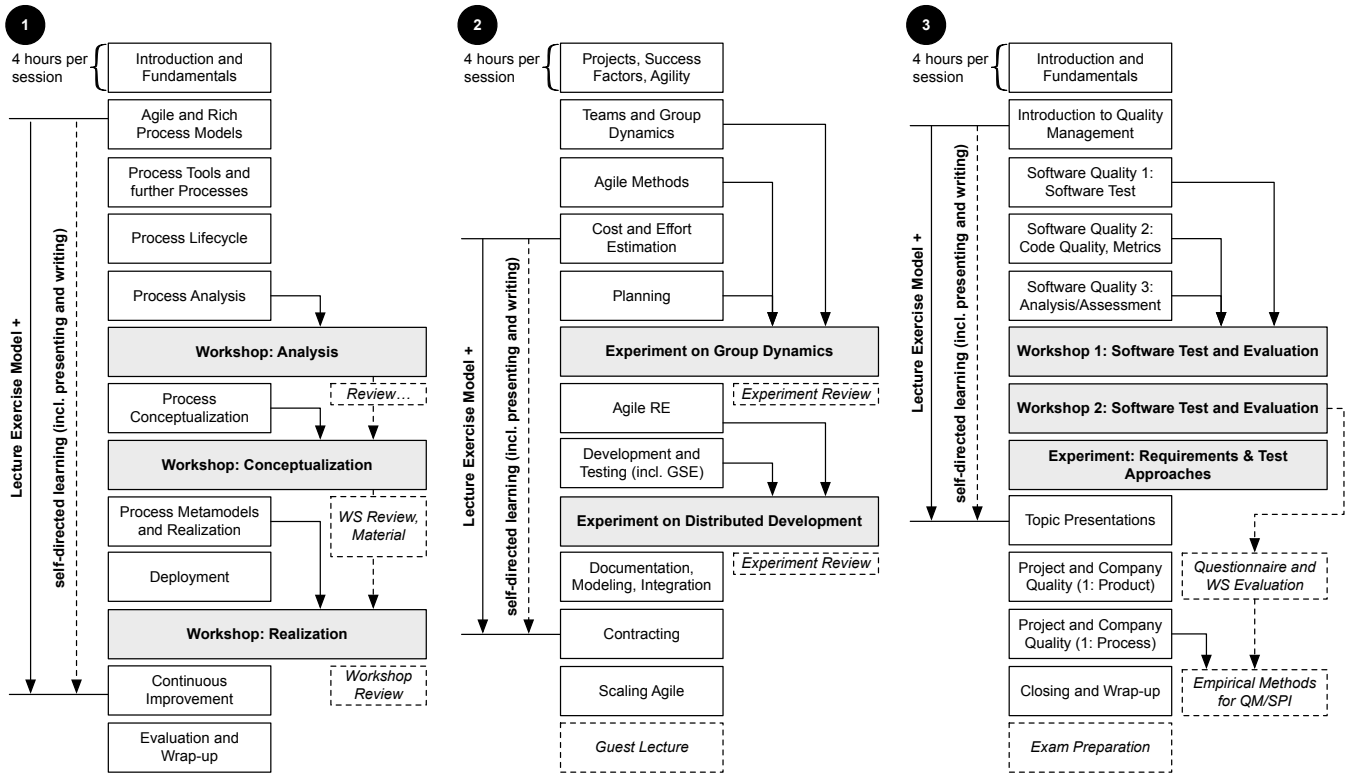

Fig. 3. Overview of the example implementations of our teaching approach: (1) Software Engineering Processes, (2) Agile Project Management and Software Development, and (3) Software Quality Management.

wrapped up, experiments in the context of the overall course are reflected, and still missing analyses are performed.

So far, we implemented advanced courses at the Master's level following this approach. That is, students attending a course following this approach should already have some basic knowledge of the topic (Section 3). In particular, our approach addresses subjects that require a certain amount of creativity and practical work, e.g., software design, software test, and software management. Furthermore, our approach primarily addresses small to medium-sized courses in which up to 25 students are enrolled.

B. Example Courses Implementing the Experimentation-based Teaching Approach

Figure 3 provides an overview of the three example implementations in which we implemented different experiments to enrich the teaching environment, and Table I provides a summary of these courses.

The courses were implemented as weekly 4-hour blocks and comprised 13 sessions each. Figure 3 shows the general course organization, including the general timeline, course units (topics), the implementation of workshops and/or experiments and their dependencies to specific course units. Furthermore, for each course, the phases of classic lecturing and more self-directed learning phases are highlighted. The figure also shows how we tailored the approach from Fig. 2 according to the particular courses, i.e., how experimentation-based course units were integrated in the overall course layout. The two courses SEP and APM are advanced courses requiring the students having successfully passed a basic project management course. The APM and SQM courses require the students to have profound knowledge in software development practices, i.e., coding and (optionally) software testing. As a "specialty", the SEP course was implemented twice: the first edition was organized using a classic course model, i.e., lectures and exercises, while the second edition was implemented using experiments [16], [18]. We discuss this in the evaluation of our teaching approach in Section IV-A1.

IV. Evaluation and Lessons Learned

Having briefly introduced our approach to implement experiments in teaching, in the following, we focus on the concept's evaluation, and we present experiences and lessons learned. For this, Section IV-A summarizes the results of the students' evaluations of the three courses. Section IV-B provides a structured collection of lessons learned and, finally, Section IV-C gives teachers a practical guideline to implement experiments in their own courses.

A. Evaluation

Since 2010, we use the same instrument for evaluating the courses. The instrument is a set questionnaires that comprises a formal course evaluation (quantitative) and another "1-minute-paper" like evaluation (qualitative). Due to space limitations,

TABLE II
FORMAL EVALUATION OF THE THREE EXAMPLE COURSES (SEE TABLE I FOR COURSE DETAILS).

Criterion	SEP 2011	APM 2012	SQM 2015
Number of questionnaires	8	15	6
Common criteria (1 = very high, 3 = just right, 5 = very low)			
Complexity	2.38	3.27	2.17
Volume	2.12	2.93	2.50
Speed	2.75	3.07	2.83
Appropriateness (effort)	3.00 (fair)	3.07 (fair)	1.67 (good)
Overall rating (1 = very good, 3 = fair, 5 = very bad)			
Lecture	1.50	1.33	1.50
Exercise	1.33	1.40	2.00
Relation to practice	1.62	1.57	2.00

we omit the full instrument, which can however be taken from our previously published papers [14], [16], [18]. In the following, we provide an aggregation of the findings from the three courses presented above.

1) Quantitative Course Evaluation: Table II summarizes the quantitative evaluation of the three example courses as provided by the students. We use the simple average of all individual ratings to find a "grade". The numbers in Table II show all courses balanced (with a tendency towards *high*) regarding complexity, volume, and speed. All courses are rated fair to good concerning the appropriateness of effort compared to the ECTS points students earn for the courses. The overall ratings of lecture and exercise parts are good to very good. Moreover, students acknowledge the relation to practice, which is also rated good (tending towards very good). While Table II presents the numbers for the three courses that have been implemented using experiments, Table III compares the two instances of the SEP course of which the first instance was implemented in the "old-fashioned" lecture-exercise model and the second instance used experiments [16], [18]. Therefore, we can compare the ratings of the two course instances, and we look for trends, e.g., for improvements regarding course quality and relevance. This comparison in Table III shows an improvement, notably concerning the exercise parts, which were considered significantly better in the experiment-based teaching model. Moreover, the ratings improved even though the course was more demanding (more student involvement, more (self-directed) work, and more deliverables to be handed in by the students).

Summarizing the results of the quantitative rating, so far, we conclude that using experiments in software engineering courses helps developing well-balanced courses. Furthermore, course-integrated experiments seem to be adequate for challenging students and providing them with the opportunity to do some "real" work. At the same time, students did not feel overstrained by the amount of theoretical knowledge and the complementing activities.

2) Qualitative Course Evaluation: For the informal evaluation, we used an extended *1-minute-paper*, i.e., students had to write down their thoughts very quickly in short statements.

TABLE III
FORMAL EVALUATION OF TWO EDITIONS OF THE SEP COURSE (COMPARISON OF WINTER SEMESTER 2010/2011 AND 2011/2012).

Criterion	2010	2011	Result
Number of questionnaires	6	8	
Common criteria (1 = very high, 3 = just right, 5 = very low)			
Complexity	3.00[a]	2.38	-0.62 ↑
Volume	2.83[b]	2.12	-0.71 ↑
Speed		2.75	-0.08 →
Appropriateness (effort)	n.a.	3.00 (fair)	n.a.
Overall rating (1 = very good, 3 = fair, 5 = very bad)			
Lecture	1.25	1.50	+0.25 ↘
Exercise	2.17	1.33	-0.84 ↑
Relation to practice	2.00	1.62	-0.38 ↗

[a] In the old questionnaire, this criterion was named "level".
[b] In the old questionnaire, "volume" and "speed" were covered by only one criterion.

TABLE IV
RESULTS FROM THE INFORMATION EVALUATION USING ONE-MINUTE-PAPERS (CONSOLIDATED SUMMARY OF ALL COURSES).

Rank	Points considered positive are:
1	Structure of the topics and the class
2	Combination of theory and practice
3	Projects in teams (atmosphere)
4	Self-motivation (due to presentations)
5	Self-directed work (group tasks, presentations)
6	Small in-class assignments
7	Mixture of tasks (reading, coding, writing, etc.)
	Points considered negative are:
1	Tough schedule
2	Tailoring of the tasks not always optimal
3	Work load (individual/group assignments)

Table IV provides a consolidated summary of the informal feedback collected in the different courses. Aspects mentioned (continuously) positive are the course structure and the mixture of activities. Due to the integration of the experiments, the course organization needs to be prepared accordingly. This leads to an early and very precise definition of course topics and course units, which give the courses a clear structure. Students welcomed this structure as it was always clear what the current topic is and how a specific topic relates to other course topics (see Fig. 3). The positively perceived mixture of the activities comes along with the self-directed learning phases. Students got their "personal" topics assigned for which they had to prepare presentations and essays, i.e., those topics had to be elaborated on a more theoretical basis and included reading, presentation, and writing tasks. Complementing, students had a variety of practical tasks (e.g., analysis, modeling, and coding). Structure and activity mix made the courses interesting and, due to the continuous practical work, the relation of topics to practical considerations was easier to communicate and understand.

On the downside, students consider the course scheduling tough and the work load high. For instance in the SQM course, one student mentioned the size of the individual assignments

too large (specifically: "*everything requiring more than 4 hours of dedicated work is a small project [..] rather than an assignment*"). Further points concern the self-directed learning phases and the examination procedures: For instance, at several occasions, students felt uncomfortable with the individual topics for the self-directed learning phases due to the more general and explorative topic description. For instance, a topic could be: present a standard (e.g., for quality management or for process maturity determination), explain what this standard is about, its strengths and weaknesses, *and* find yourself some good examples to explain selected aspects. While some students did not like this kind of open task, others welcomed exactly this option. Another critical point is the exam. Students are paid with ECTS-point-weighted grades, and there should be a fair revenue for the effort spent on a course. Therefore, we also started using a cumulative grading model, i.e., students receive a certain amount of points for the respective tasks and these points contribute to the final grade. Most of the students welcomed this procedure, since it allows for continuously working towards a good grade. Yet, some students signed off and left the courses because of this grading model, which they considered too demanding.

B. Discussion and Lessons Learned

In [16], we reported initial lessons learned from the pilot implementation (SEP course, cf. Table III), which were continuously extended by our other previously published papers. In this section, we consolidate and structure our lessons learned across all courses.

a) Improvement of general communication skills: Under communication skills, we subsume reading, writing, and presentation skills. In the example courses, students had to explore topics themselves, and they had to present their findings to their class mates. This includes, inter alia, finding and understanding literature "beyond Google", giving presentations, and writing. For most of the students, our courses were the first occasion in which they had to combine all these activities. Yet, proper training can bring students to a professional level quickly, as for instance demonstrated by a publication [48], which emerged from a follow-up activity of the SEP course.

b) Improvement of collaboration skills: If students are trained in a more individual learning pattern, experiments allow for teamwork, e.g., when working on specific topics or in assignments. Collaboration can be further intensified, if the experiments involve group work [17] or aim at experiencing collaboration [7], [20]. Especially the latter aspect is a rare opportunity for students to experience difficult or even critical situations.

c) Improvement of individual problem-solving skills: If students grew up in a more team- or project-based learning environment, experiments allow for training individual problem-solving skills (for instance in the SQM course in which experiments were implemented that students had to carry out individually, e.g., [32]).

d) Creating experts: Experiments allow students to dig into the details of specific topics. Complementing the general education provided by the common curriculum, students have the opportunity to intensively work on a specific topic and to develop expertise. So far, several students from the different courses found topics of interest and continued working on these topics, e.g., in study projects [47]–[49], in-company periods, and after graduation.

e) No difference between real-world and artificial tasks: A conclusion we could draw from running the courses is that it is not always necessary to ground course activities in real-life projects. We agree that the best presentation of practical issues is using real cases—and it is also an important selling point to explicate the relevance of a course. However, for some topics, real cases are not the optimal solution (because they are too big) or even counter-productive, as we explained for instance for project management [7], since real projects bear the risk to shift the focus away from the originally targeted problems. Experiments as for instance implemented in [7] or [20] provide a better scope. Therefore, the learning is to critically select cases of interest and, quite often, an artificial setup, i.e., a small controlled in-class experiment, is enough.

f) High work load: The first thing students mentioned was the high work load. Yet, this perceived high work load emerges from the variety of challenges the students faced. In particular, students took over responsibility for topics and had to work in a self-directed manner—if not available, they had to learn required skills on the fly with teachers providing guidance. Furthermore, due to the structure of the course, the (often hidden) volume and complexity of a course becomes explicit, and students go through all these topics intensively. Continuous participation in classes is required—the "consume-lecture-only" approach does not work anymore. So far, we mostly experienced a boost in students' performance, and even though the students mention the high(er) workload, the formal evaluation still shows good grades for the courses (cf. Table II).

g) Demanding preparation for teachers: Implementing in-class experiments and aligning the experiments with the theoretical concepts to be taught causes effort. For instance, for the real-world example for the SEP course [18], it was necessary to identify a proper case and to tailor it for the course context, i.e., to reduce the example and to pre-digest it to emphasize the course-relevant aspects. For the experiments in the APM course, e.g., [7], it was necessary to develop the idea and a sound experiment design. Finally, for the experiment used in the SQM course, partners had to be identified, experiment setups, e.g., [32], had to be analyzed and adapted, and so forth. This makes the preparation demanding, yet, it also offers the chance to implement teaching-related collaboration. However, once a setup was defined, over time, reuse is possible and reduces the effort. To support other teachers, Section IV-C provides some practical advice.

h) Risk management is necessary: There is also one major risk to be considered: giving the students responsibility to work on and present specific topics does not release the teacher from preparing fall-back sessions. It might always

TABLE V
RECOMMENDATIONS FOR THE COURSE ORGANIZATION.

Tip	Description
Prerequisites	Teachers need sufficient time for preparation. In addition to classic supporting material, e.g., scripts or slides, teachers also have to prepare the artifacts for the experiments such as: • Project assignments for the activities, e.g., 1-page task descriptions • All input artifacts required for the activities, e.g., experimentation kits or data • Surveys and other evaluation-related instruments • Analysis report(s), in case the activities cannot be analyzed/evaluated in real time It needs to be ensured that the required background knowledge is in place. That is, if students carry out experiments, before conducting the actual activity, it is recommended to have different smaller exercises to prepare the students accordingly, i.e., exercise the way of work or exercise similar cases to develop context knowledge. If students shall conduct an experiment themselves, e.g., a case study with an integrated data analysis, it is required to also train the required empirical methods. For instance, if students conduct an experiment on software quality that includes some sort of user survey, knowledge about developing proper questionnaires and how to analyze them is imperative.
Meetings	In case, assignments are provided that span the whole semester, teachers need to allocate time for meetings. Yet, such meetings cannot be organized like a consultation hour for a regular course—those meetings have to be considered status and working meetings similar to real projects. To train the students, it has been proven to hand over the organization to the respective student groups.
Examination	Since a course using experiments as teaching tool is not done "as usual", the examination procedure needs to be adapted. Special attention should be payed to the possibilities of a procedure that allows for a continuous evaluation of the students' performance. That is, there is not only one appointment where students have to be in great shape, as students' work continuously over the whole semester for their final grades. For small and medium groups (up to 25 students), the following procedure worked well: • Students prepare presentations on their topics. • Students write essays on their selected topics. • In case a semester-spanning assignment was given, exam-relevant deliverables have to be defined, e.g., documentation, data, or software. • In case the different activities were spread across the semester, an oral exam is the preferred way, since it allows for testing basic knowledge as well as running a discussion on in-depth topics. The different partial/tentative deliverables from the course activities are considered for finding the final grade. The final grade can be found by weighting the different parts, e.g., 1/3 for the presentations and essays, and 2/3 for the oral exam.

happen that students get ill, drop out, or just do not do their tasks. For such situations, teachers must be able to take over, i.e., using experiments does not come along with a net reduction in preparation effort.

i) Resource consumption: Experiments do not only challenge the students, but also challenge teachers as a close collaboration is required. For instance, while students work on their specific or new topics, teachers guide the students and review their (tentative) results. Furthermore, courses that use experiments are not static, i.e., they require the teacher to bring in some flexibility, e.g., taking tentative results from the students and consolidate them for the next phase. For instance, in [17], it was necessary collecting results and creating a consolidated artifact, such that the students could continue. This puts a limit on the course size as teachers need to manage this effort. Experience gathered so far suggests classes of about 25 students as upper bound for *direct* interaction. In [50], we presented an adaptation for larger classes and used a *proxy-based interaction* pattern, which uses collaborating expert teams. Still, the resource consumption is high, yet, we could demonstrate how to scale our teaching approach.

j) Not feasible for undergrads: All example courses were implemented in Master programs. For testing the applicability to undergraduate teaching, the experiment from the SQM course was partially implemented twice in a second-year Bachelor course. Experience from these two course instances suggests that experiment-based teaching is only partially applicable to undergrad students. Organizing selected lab sessions using experiments showed beneficial; especially for problem understanding and solution development. Yet, undergrads require a much more extensive preparation for experiments and the required work pattern. Hence, applied to Master courses, experiments showed several advantages whereas this approach currently seems to be too demanding for undergrads.

k) General learning: Additionally to the above lessons learned, an overall learning is concerned with transparency for the students. This means that the overall course organization, the requirements regarding the course activities, and also the examination procedures need to be made explicit from the very beginning on. Most students are unfamiliar with this kind of course organization and, thus, might be reluctant as their concern is the potential impact to the final grades and the effort required to pass the course.

C. Course Organization and Administration

Organization and administration of experiments used in courses need to be done carefully as it requires some effort to run experimentation-based courses. Additionally, in [14], we

present a guideline which type of empirical study is feasible for a specific context, e.g., is it better to use a simulation or a controlled experiment to achieve certain course goals and what are the challenges and possible outcomes. For experiments as used in our teaching approach, Table V summarizes the key points to be taken into account.

Another aspect is crucial: In the course syllabus—at latest in the course-opening session—students need to be informed about the way the course is run. Experience shows most of the students fairly unprepared for this way of working in a course, since it completely contradicts a "consume-lecture-only" participation approach. This is also shown by the course evaluations in Section IV-A: students consider this format beneficial yet challenging. Therefore, transparency and openness right from the start on are required.

V. Conclusion

In this paper, we report our experiences regarding the use of experiments as teaching tool in software engineering courses. The purpose of using experiments is to provide a means to systematically combine theoretical knowledge and practical exercises. Using in-class experiments is beneficial to students as they can exercise a variety of activities in combination, and students can also steer a course themselves by setting own focal points. Teachers benefit from the non-static course organization. For instance, hot topics of the fields can be easily brought into the course as new activity and course contents can also be organized around teachers' research.

The evaluation shows the three example courses challenging the students, and the evaluation shows that students perceived the courses of high quality and relevance. A comparative analysis of the course *Software Engineering Processes*, which was offered once in the classic format and once using experiments, showed a significant increase of the student rating, notably regarding the exercise parts—it also needs to be mentioned that the final grades improved in the second instance of the course.

Our lessons learned include that using experiments provides students the opportunity to improve their communication and collaboration skills. Furthermore, students have the opportunity to build expertise in fields of interest. However, our lessons learned also show this teaching format challenging for the teachers as it requires careful planning and preparation, risk management, and an open attitude towards flexibility. Also, due to the close collaboration, applying this teaching format limits the class size. However, once the course designs are in place, reuse is possible.

Acknowledgements

We would like to thank all our colleagues, who helped us implementing the different courses in the past years. We also like to thank all the students, who accepted the challenge of participating these courses and who provided us with valuable feedback. Finally, we owe thank to the Faculty of Informatics of the Technical University of Munich that provided financial support as part of the "Ernst Otto Fischer Teaching Award" (2012, http://portal.mytum.de/studium-und-lehre/lehrpreise/ernst_otto_fischer_lehrpreis.html).

References

[1] M. Ardis, G. Hislop, M. Sebern, D. Budgen, J. Offutt, and W. Visser, Eds., *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. ACM, IEEE, February 2015.

[2] A. Pyster, Ed., *Curriculum Guidelines for Graduate Degree Programs in Software Engineering*. Stevens Institute of Technology (in cooperation with ACM), September 2009.

[3] P. Bourque and R. E. Fairly, Eds., *SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2014, no. ISBN-13: 978-0-7695-5166-1.

[4] T. Clear, S. Beecham, J. Barr, M. Daniels, R. McDermott, M. Oudshoorn, A. Savickaite, and J. Noll, "Challenges and recommendations for the design and conduct of global software engineering courses: A systematic review," in *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education (Working Group Reports)*, ser. ITICSE-WGR. New York, NY, USA: ACM, 2015, pp. 1–39. [Online]. Available: http://doi.acm.org/10.1145/2858796.2858797

[5] W. S. Humphrey, M. D. Konrad, J. W. Over, and W. C. Peterson, "Future directions in process improvement," *Crosstalk – The Journal of Defense Software Engineering*, vol. 20, no. 2, pp. 17–22, February 2007.

[6] B. Brügge, S. Krusche, and L. Alperowitz, "Software engineering project courses with industrial clients," *Trans. Comput. Educ.*, vol. 15, no. 4, pp. 17:1–17:31, December 2015. [Online]. Available: http://doi.acm.org/10.1145/2732155

[7] M. Kuhrmann and J. Münch, "When teams go crazy: An environment to experience group dynamics in software project management courses," in *International Conference on Software Engineering*, ser. ICSE. New York, NY, USA: ACM, May 2016, pp. 412–421.

[8] H.-L. Yang and J.-H. Tang, "Team structure and team performance in is development: A social network perspective," *Inf. Manage.*, vol. 41, no. 3, pp. 335–349, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1016/S0378-7206(03)00078-8

[9] J. H. Bradley and F. J. Hebert, "The effect of personality type on team performance," *Journal of Management Development*, vol. 16, no. 5, pp. 337–353, 1997.

[10] N. Gorla and Y. W. Lam, "Who should work with whom?: Building effective software project teams," *Communications of the ACM*, vol. 47, no. 6, pp. 79–82, June 2004.

[11] G. P. Sudhakar, A. Farooq, and S. Patnaik, "Soft factors affecting the performance of software development teams," *Team Performance Management: An International Journal*, vol. 17, no. 3/4, pp. 187–205, 2011.

[12] F. Fagerholm, M. Ikonen, P. Kettunen, J. Münch, V. Roto, and P. Abrahamsson, "Performance alignment work: How software developers experience the continuous adaptation of team performance in lean and agile environments," *Information and Software Technology*, vol. 64, pp. 132–147, August 2015.

[13] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A., *Experimentation in Software Engineering*. Springer, 2012.

[14] F. Fagerholm, M. Kuhrmann, and J. Münch, "Guidelines for using empirical studies in software engineering education," *PeerJ Computer Science*, vol. 3, no. e131, September 2017.

[15] A. Rösel, *Are We Ready for Disruptive Improvement?* Cham: Springer International Publishing, 2016, pp. 77–91. [Online]. Available: https://doi.org/10.1007/978-3-319-31545-4_5

[16] M. Kuhrmann, "A practical approach to align research with master's level courses," in *International Conference on Computational Science and Engineering*, ser. CSE. Washington, DC, USA: IEEE, December 2012, pp. 202–208.

[17] M. Kuhrmann, D. M. Fernández, and A. Knapp, "Who cares about software process modelling? a first investigation about the perceived value of process engineering and process consumption," in *International Conference on Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 2013, vol. 7983, pp. 138–152. [Online]. Available: https://www.e-proof.sps.co.in/LNCS/request.asp?file=ihejjbijdbdi

[18] M. Kuhrmann, D. M. Fernández, and J. Münch, "Teaching software process modeling," in *International Conference on Software Engineering*, ser. ICSE. Washington, DC, USA: IEEE, 2013, pp. 1138–1147.

[19] M. Kuhrmann, H. Femmer, and J. Eckhardt, *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills*. IGI Global, 2014, ch. Controlled Experiments as Means to Teach Soft Skills in Software Engineering, pp. 180–197.

[20] M. Kuhrmann and J. Münch, "Distributed software development with one hand tied behind the back: A course unit to experience the role of communication in gsd," in *1st Workshop on Global Software Engineering Education (in conjunction with ICGSE'2016)*, ser. GSE-Ed (ICGSEW). Washington, DC, USA: IEEE, 2016, pp. 25–30.

[21] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *Transactions on Software Engineering*, vol. 12, no. 7, pp. 733–743, July 1986.

[22] P. Runeson, "Using students as experiment subjects–an analysis on graduate and freshmen student data," in *International Conference on Empirical Assessment in Software Engineering*, 2003, pp. 95–102.

[23] E. Kamsties and C. M. Lott, "An empirical evaluation of three defect-detection techniques," in *European Software Engineering Conference*. London, UK: Springer, 1995, pp. 362–383.

[24] E. Keenan, A. Steele, and X. Jia, "Simulating global software development in a course environment," in *International Conference on Global Software Engineering*, ser. ICGSE. Washington, DC, USA: IEEE, 2010, pp. 201–205.

[25] I. Richardson, A. E. Milewski, N. Mullick, and P. Keil, "Distributed development: an education perspective on the global studio project," in *International Conference on Software Engineering*, ser. ICSE. New York, NY, USA: ACM, 2006, pp. 679–684.

[26] S. Beecham, T. Clear, J. Barr, M. Daniels, M. Oudshoorn, and J. Noll, "Preparing tomorrow's software engineers for work in a global environment," *IEEE Software*, vol. 34, no. 1, pp. 9–12, Jan 2017.

[27] Huang, L., Dai, L., Guo, B., and Lei, G., "Project-driven teaching model for software project management course," in *Proceedings of the International Conference on Computer Science and Software Engineering*. Washington, DC, USA: IEEE, December 2008, pp. 503–506.

[28] D. Dahiya, "Teaching software engineering: A practical approach," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 2, pp. 1–5, March 2010.

[29] G. Bavota, A. D. Lucia, F. Fasano, R. Oliveto, and C. Zottoli, "Teaching software enginerring and software project management: An integrated and practical approach," in *International Conference on Software Engineering*, ser. ICSE. Washington, DC, USA: IEEE, June 2012, pp. 1155–1164.

[30] W. Pádua, "Measuring complexity, effectiveness and efficiency in software course projects," in *Proceedings of the International Conference on Software Engineering*, ser. ICSE. New York, NY, USA: ACM, May 2010, pp. 545–554.

[31] R. C. Green and J. T. Chao, "Ten years of the agile software factory for software engineering education and training," in *IEEE Conference on Software Engineering Education and Training*, ser. CSEE&T. Washington, DC, USA: IEEE, Nov 2017, pp. 182–186.

[32] D. Fucci and B. Turhan, "A replicated experiment on the effectiveness of test-first development," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM. New York, NY, USA: ACM, October 2013, pp. 103–112.

[33] D. Fucci, B. Turhan, and M. Oivo, "Impact of process conformance on the effects of test-driven development," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 10:1–10:10. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652526

[37] J. Dillon, "A Review of the Research on Practical Work in School Science," King's College, Tech. Rep., 2008. [Online]. Available: http://score-education.org/media/3671/review_of_research.pdf

[34] ——, "On the effects of programming and testing skills on external quality and productivity in a test-driven development context," in *International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE. New York, NY, USA: ACM, 2015, pp. 25:1–25:6. [Online]. Available: http://doi.acm.org/10.1145/2745802.2745826

[35] S. Ball, T. Emerson, J. Lewis, and J. T. Swarthout, "Classroom experiments," Available from http://serc.carleton.edu/sp/library/experiments/index.html, May 2012.

[36] J. Parker. (1995) Using laboratory experiments to teach introductory economics. Online http://academic.reed.edu/economics/parker/ExpBook95.pdf.

[38] C. Wohlin, "Empirical software engineering: Teaching methods and conducting studies," in *Proceedings of the International Workshop on Empirical Software Engineering Issues: Critical Assessment and Future Directions*, ser. LNCS. Springer, 2007, vol. 4336, pp. 135–142. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71301-2_42

[39] E. Dale, *Audiovisual methods in teaching*, 3rd ed. Dryden Press, 1969.

[40] P. Brereton, M. Gumbley, and S. Lees, "Distributed student projects in software engineering," in *Conference on Software Engineering Education and Training*, ser. CSEE&T. Washington, DC, USA: IEEE, February 1998, pp. 4–15.

[41] I. Dukovska-Popovska, V. Hove-Madsen, and K. B. Nielsen, "Teaching lean thinking through game: Some challenges," in *Proceedings of the European Society for Engineering Education on Quality Assessment, Employability & Innovation*, ser. SEFI. Sense Publishers, July 2008.

[42] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch, "Building blocks for continuous experimentation," in *International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE. New York, NY, USA: ACM, 2014, pp. 26–35.

[43] J. Münch, F. Fagerholm, P. Johnson, J. Pirttilahti, J. Torkkel, and J. Järvinen, "Creating minimum viable products in industry-academia collaborations," in *Lean Enterprise Software and Systems Conference*, ser. LESS. Berlin and Heidelberg: Springer, 2013, pp. 137–151.

[44] Münch, J., Pfahl, D., and Rus, I., "Virtual software engineering laboratories in support of trade-off analyses," *International Software Quality Journal*, vol. 13, no. 4, 2005.

[45] Mandl-Striegl itz, P., "How to successfully use software project simulation for educating software project managers," in *Frontiers in Education Conference*. Washington, DC, USA: IEEE, October 2001, pp. 19–24.

[46] C. Deiters, C. Herrmann, R. Hildebrandt, E. Knauss, M. Kuhrmann, A. Rausch, B. Rumpe, and K. Schneider, "Glose-lab: Teaching global software engineering," in *International Conference on Global Software Engineering*, ser. ICGSE. Washington, DC, USA: IEEE, August 2011, pp. 156–160.

[47] A.-D. Rein and J. Münch, "Feature prioritization based on mock purchase: A mobile case study," in *Lean Enterprise Software and Systems Conference*, ser. LESS. Berlin Heidelberg: Springer, 2013, pp. 165–179.

[48] M. Kuhrmann, D. M. Fernández, and R. Steenweg, "Systematic software process development: Where do we stand today?" in *Proceedings of the International Conference on Software and Systems Process*, ser. ICSSP. New York, NY, USA: ACM Press, 2013, pp. 166–170.

[49] J. W. Jacobsen, M. Kuhrmann, J. Münch, P. Diebold, and M. Felderer, *On the Role of Software Quality Management in Software Process Improvement*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, vol. 10027, pp. 327–343. [Online]. Available: https://doi.org/10.1007/978-3-319-49094-6_21

[50] M. Kuhrmann, "Teaching empirical software engineering using expert teams," in *Software Engineering im Unterricht der Hochschulen*, ser. SEUH, vol. 1790. CEUR Workshop Proceedings, February 2017, pp. 20–31.