Using Mini-Projects to Teach Empirical Software Engineering

Michael Felderer[1] and Marco Kuhrmann[2]

[1]University of Innsbruck, michael.felderer@uibk.ac.at
[2]Clausthal University of Technology, kuhrmann@acm.org

Abstract

Empirical studies have become a central element of software engineering research and practice. Yet, teaching the instruments of empirical software engineering is challenging, since students need to understand the theory of the scientific method and also have to develop an understanding of the application of those instruments and their benefits. In this paper, we present and evaluate an approach to teach empirical software engineering with course-integrated mini-projects. In mini-projects, students conduct small empirical studies, e.g., surveys, literature reviews, controlled experiments, and data mining studies in collaborating teams. We present the approach through two implementations at two universities as a self-contained course on empirical software engineering and as part of an advanced software engineering course; with 101 graduate students in total. Our evaluation shows a positive learning experience and an improved understanding of the concepts taught. More than a half of the students consider empirical studies helpful for their later careers. Finally, a qualitative coding and a statistical analysis showed the proposed approach beneficial, but also revealed challenges of the scientific work process, e.g., data collection activities that were underestimated by the students.

1 Introduction

Empirical software engineering aims at making software engineering claims measurable, i.e., to analyze and understand phenomena in software engineering and to evaluate software engineering approaches and solutions, and to ground decision-making processes in evidence. For this, an extensive portfolio of instruments for empirical software engineering has been developed. For instance, Wohlin et al. [27] provide a collection of instruments, e.g., controlled experiments, surveys and case studies, to be used for empirical studies in software engineering. Kitchenham et al. [13] extended these instruments by a detailed guideline for conducting systematic reviews. For most of the basic instruments used in empirical software engineering today, extended and more detailed (pragmatic) guidelines exist, such as for systematic reviews [16, 28], systematic mapping studies [23, 24], multi-vocal reviews [10, 11], or surveys [12, 21]. All these instruments are meant to support researchers and practitioners alike in conducting empirical studies and to ground their work and decisions in evidence.

Conducting empirical studies is challenging and requires careful preparation and a disciplined work approach. Quite often, students consider empirical studies of little to no help when it comes to software development and project work, since the relation to actual development tasks is not obvious. Yet, many of today's applications rely on data, e.g., machine learning systems like text and speech recognition, IoT devices, and autonomous cars. Empirical methods as such are about data analysis and, thus, provide a suitable approach to teach data analysis—or data engineering in general—that is a core competence in data-intensive applications. Furthermore, modern software development paradigms, such as DevOps including continuous integration and deployment, utilize data, e.g., to analyze a system's performance, to predict defects, and to make informed decisions in the development process as practiced in continuous experimentation [5]. Therefore, it is necessary for teachers to open the students' minds for a rigorous and evidence-based work approach.

In this paper, we present and evaluate the concept of *course-integrated mini-projects* to teach empirical software engineering instruments. Our approach helps students learn how to conduct empirical studies and understand the instruments and challenges coming along with such studies. Collaborating project teams conducting small empirical studies form the basis of our approach. We implemented the approach in two courses at the *University of Southern Denmark* (2016, 68 students) and *University of Innsbruck* (2017, 33 students). Mini-projects allow students to learn empirical instruments by practically applying them. Our evaluation shows a positive learning experience and an improved understanding of (empirical) software engineering concepts. More than half of the students perceive empirical studies helpful for their later ca-

reers. Our evaluation also shows that notably data collection activities (e.g., for surveys and experiments) are underestimated by the students. Our findings thus lay the foundation for improving research-oriented courses that require data and data analysis.

The remainder of the paper is organized as follows: Section 2 gives an overview of background and related work. Section 3 describes the mini-project approach, and Section 4 presents the approach's evaluation based on two implementations at the *University of Southern Denmark* and the *University of Innsbruck* respectively. We conclude the paper and discuss future work in Section 5.

2 Background and Related Work

Using empirical studies in software engineering education is not a new idea [2]. However, empirical studies—notably (controlled) experiments—are mainly used as a tool to support research using students as subjects, but got little appreciation as a teaching tool in software engineering in the first place. That is, students only get in touch with empirical studies as subjects in an empirical inquiry, and they have to carry out tasks, e.g., in an experiment as for instance reported in [7, 8, 18, 20]. Yet, teaching empirical software engineering as a subject requires a setup in which empirical studies are the main subject or at least provide a significant contribution to a course. In this regard, Wohlin [26] proposes three levels for integrating empirical studies in software engineering courses: (i) integration in software engineering courses, (ii) as a separate course, and (iii) as part of a research method course. Wohlin mentions that introducing empirical software engineering will provide more opportunities to conduct empirical studies in student settings, but that educational and research objectives need to be carefully balanced. Dillon [4] comes to the same conclusion and considers a successful observation of a phenomenon as part of an empirical study not be an end in itself. Students need time to get familiar with ideas and concepts associated with the phenomenon under observation. Finally, Parker [22] considers experiments distinctive and more participative. Students are likely to remember lessons associated with experiments.

In this paper, we present an approach that considers empirical studies major subjects of a course and that uses such studies as teaching tool. Referring to established learning models such as Bloom's *Taxonomy of Learning* [1] and Dale's *Cone of Learning* [3], we aim to include as many *active learning* parts as possible in the courses. Still, we use *passive learning* methods to transfer knowledge about theoretical basics, such as methods and their application contexts. Addressing the *active learning* levels, however, is challenging. In "ordinary" software engineering education, project courses are used to train software project work. For empirical studies, it is required that the students carry out actual research to practice the application of the empirical instruments. In our previous work [17–19], we presented different, self-contained classroom experiments and developed a guideline to select the best-fitting study type for a specific context [6]. In [14], we introduced a teaming model that helps implementing empirical studies in larger project courses.

We contribute a generalized concept grounded in [6, 14] that allows for including empirical studies as course units. We implemented and evaluated our approach in a course on empirical software engineering and an advanced course on software engineering and demonstrate how to implement and scale course-integrated empirical studies.

3 Course-Integrated Mini-Projects

We present the *course-integrated mini-projects* approach in Section 3.1. The presentation includes the description of the team setups, project and task descriptions, and examples for which we present details in Section 3.2. Section 3.3 demonstrates two integration strategies: the first integration strategy is a self-contained course on empirical software engineering [14] and the second strategy is a topic-specific part of an advanced software engineering course.

3.1 Mini-Projects and Project Teams

Figure 1 shows the general organization model for the mini-project approach. A `MiniProject` has a `Topic`, a `Schedule`, and optional `Reference Literature` and `Input Data`. It is always carried out using at least one `Method`, e.g., an experiment [27], a case study [25], and a survey [21]. Finally, every mini-project consists of a `Project Team` and an `Advisory Team`, which we describe in more detail in the following.

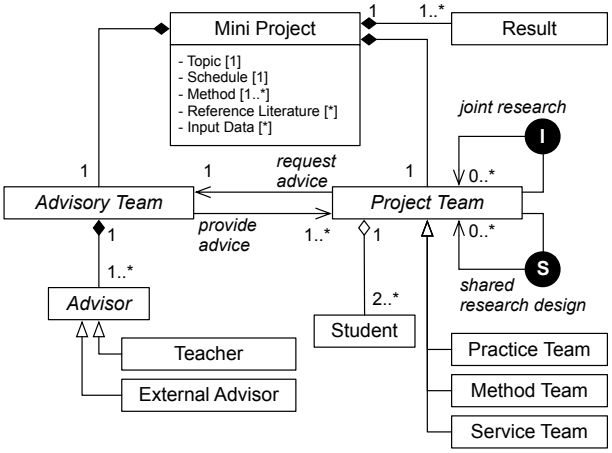

Figure 1: Overall organization model of mini-projects.

Advisory Teams An advisory team bundles all advisors involved in a specific mini-project—usually one or two persons. An `Advisor` is either the teacher of the course or an external advisor, such as an external

Style	Description
Isolated	The "normal" way of doing a mini-project is the *isolated* way of working. Isolated means that a project team has a self-contained task that can be worked on without any interaction with other teams.
Joint	This style is applied if project teams *collaboratively* work on a *joint (research) project*. A complex project is broken down into a number of smaller projects. Project teams thus have to be coordinated in terms of task distribution, scheduling, and result synthesis.
Shared	This style is applied if project teams *competitively* work on the *same (research) topic*. Two or more teams are assigned the same task; team-specific methods can be varied and results can be compared. That is, this style helps conducting controlled experiments or implementing independently conducted studies.

Table 1: Overview of the different interaction styles among mini-project teams.

project topic sponsor [14]. Besides offering and promoting project topics, advisors regularly interact with the project teams for which they handle individual support requests, and they provide general technical and methodical support.

Project Teams A project is composed of all students working on a specific problem. Project teams can interact with each other. We distinguish the three interaction styles *isolated*, *joint*, and *shared*, which are explained in Table 1. Furthermore, we distinguish three types of project teams according to the type of task they are working on: a `Practice Team` performs an "active" task, e.g., a development task or a research task. A `Method Team` deals with methodological expertise, i.e., it develops competencies regarding specific (scientific) methods and offers "consultancy services" in terms of applying a specific method "right" to practice teams. Finally, a `Service Team` develops skills in more general topics, such as data analysis or presentation, and offers respective "services" to other teams—practice teams and method teams alike.

3.2 Project- and Task Descriptions

Every mini-project is supposed to produce at least one `Result`. In this section, we provide a blueprint for a 1-page project- and task description, which also illustrates the manifestation of the different attributes of the class `Mini Project` (Figure 1).

For every project, a description that includes tasks, dates, and expected results is necessary. Table 2 provides a summary and a description of task-description items that we consider relevant. The *work description* requires special attention as it comprises the detailed activity list, input material, and the description of expected results. The expected results are specified right here; alternatively, a separate catalog of results has to be provided, e.g., including templates and mandatory/recommended outlines. Relevant types for project outcomes are, e.g., (research) data[1], essays or reports, presentations, tutorials, and software. The second important item of the task description is the list of *related projects*. For instance, if a task is a collaborative task, this list refers to all related projects that contribute to the overall project goal. Furthermore, this list also refers those method and service teams that provide useful support, e.g., if the project is concerned with developing and conducting a survey, this list can refer to a method team that is focused on the theoretical aspects of survey research. Finally, the task description can also be used to develop a checklist, which is used for the final submission. This checklist helps students to check if their delivery package is complete, and it helps teachers validating the delivery and grade its components. Figure 2 shows

Item	Description
Metadata	This section contains all information relevant to a task, e.g., hand-in date.
Title	A project needs a telling title and an ID
Context	This section briefly describes the context of the project and provides a short summary of the basic tasks. *Recommendation:* the context section should be treated as an abstract, such that it can be used as a teaser and a small piece of information, e.g., used in a course management system.
Work Description	The detailed work description contains at least: 1. A detailed task list. 2. A list of input/reference material. 3. A list of deliverables to be shipped. *Note:* The level of detail depends on the actual task, i.e., for an "explorative" task, the description needs to be more open while a specific development task requires a more detailed task description.
Schedule	The basic schedule lists all deadlines and the respectively expected results.
Related Projects	Our concept allows for collaborative and competitive work (Figure 1 and Table 1). If such a collaborative/competitive work style is implemented, this section provides the information about the other teams. If method or service teams provide useful services to a project, such teams are referred here too.
Literature	This section lists selected reference literature relevant to a project.

Table 2: Structure of a mini-project task description (recommended minimal elements).

[1] Recommendation: If students conduct a research task, analyzed data that is necessary for the project documentation must always be complemented with the original raw data.

Conduct a Survey on Practitioner Requirements in Software Testing (ID15-P)

In this project, you have to conduct a survey on the expectations of practitioners regarding software-testing work in academia, and you have to present your findings. In particular, you are expected to:

1. Prepare a survey of the practitioners organized in the **Technology Denmark** cluster
2. Carry out the survey and analyze the survey results
3. Report on the experiment and (initial) results

Work Description and Work Plan

1.1 Work Description

This practical project is a 4-student project. Starting with a given questionnaire, you carry out an interview survey among industry practitioners. In particular, you are expected to carry out the following tasks:

1. Analyze the input material provided.
2. Plan survey and schedule interviews with practitioners.
3. Carry out the study by surveying the practitioners.
4. Analyze and present your findings.

Your team will receive the following input material:

1. A guideline to conduct survey research in Software Engineering [1, 2]
2. A pre-defined questionnaire and guideline (project meeting)

Your team is expected to deliver the following items:

- An interview plan (research protocol).
- Documented (raw) research data, incl. all sorts of analysis documents and notes
- A 15-minutes presentation
- A 10-page report

1.2 Basic Schedule and Related Projects

Your project follows the following basic schedule:

Week 35: Topic assigned and work starts
Week 46: Presentation of the outcomes (15-minute talk)
Week 47: Submission

Please also consult the following projects to get further information and cross-project collaboration.

Project 01 (What is a survey?); 09 (Introduction to Data Visualization);

Project 12, 13, 16 (further survey projects)

1.3 Further Literature

The following listed publications should be regarded when working on your project:

[1] Linåker, J., Sulaman, S. M., Maiani de Mello, R., & Höst, M.: Guidelines for Conducting Surveys in Software Engineering. Technical Report, Lund University, 2015
[2] Kitchenham, B., Pfleeger, S.: Personal Opinion Surveys. *Chapter in book: Shull, F., Singer, J., and Sjoberg, D.: Guide to Advanced Empirical Software Engineering*, Springer, 2008

Figure 2: Example of a mini-project task description.

a practically used example of a task description as described in Table 2. This task description is taken from the course given at *University of Southern Denmark* and describes a survey research task, which was performed collaboratively with two external researchers [9]. This particular project was a collaborative project. Specifically, two teams (No. 15 and 16; see the *related projects* part) were assigned one task, but had to conduct the survey with different target groups. Each team submitted its own data and report, but, both teams gave only one joint presentation.

3.3 Course-Integration Strategies

We describe two implementation strategies for the presented approach using two graduate courses. For these implementations, we provide a short summary of the respective courses and their learning goals, and we provide an overview of the projects implemented in the respective courses (Table 3). Furthermore, this section lays the foundation for the approach's evaluation, which is presented in Section 4.

3.3.1 Implementation as a Self-Contained Empirical Software Engineering Course

A course on the *Scientific Method* (SCM, *University of Southern Denmark*, *SDU*, 2016) implemented the presented approach as a self-contained master-level course. Figure 3 illustrates the overall organization of the SCM course showing the introduction parts and the active learning/project parts.

The goal of the SCM course was to teach empirical software engineering as main subject by letting the students perform small studies themselves [14]. Specifi-

Study Type	SCM	Gr	Top	ASE	Gr	Top
Theory (tutorial)	✓	9	9	✗		
Experiment	✓	1	1	✓	2	1
Survey	✓	4	2	✓	3	1
Systematic Review	✓	3	2	✓	1	1
Mapping Study	✓	1	1	✗		
Simulation	✓	2	2	✗		
Data Mining Study	✗			✓	7	3

Table 3: Overview of the study types implemented including the number of groups for a specific method (Gr) and the number of topics per study type (Top).

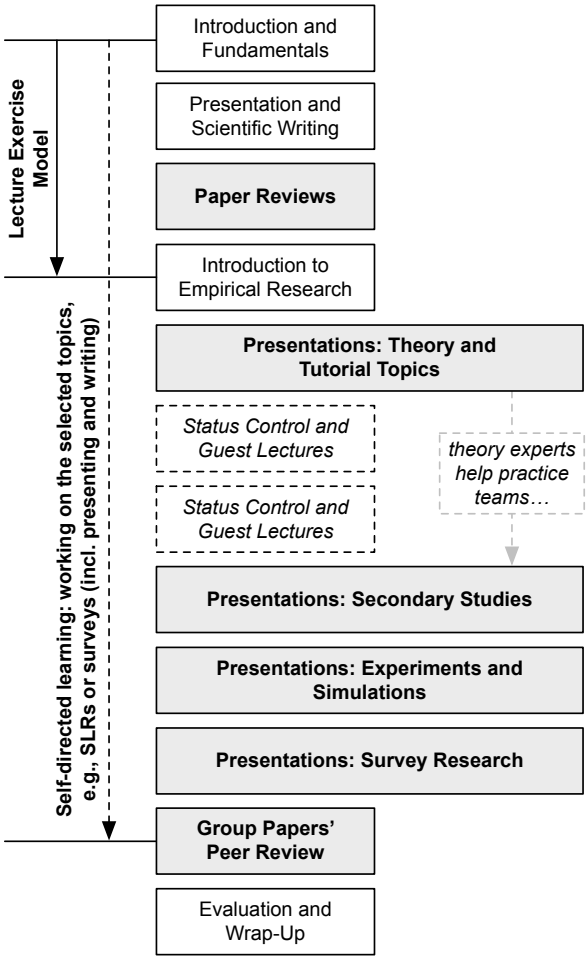

Figure 3: Overview of the topics and the general organization of the SCM course.

cally, the major learning goals of the SCM course were defined as follows:

- After the course, students know the basic terminology and the key concepts of the scientific method.
- After the course, students know and understand the most important empirical research methods.
- After the course, students have shown their ability to practically apply one research method, conduct and report on a small research project.

In total, 30 research topics were presented to the students. According to their preferences, students could apply for up to three topics, which built the foundation for the final team setup. Finally, 20 projects were started with two to three students each. Besides the theoretical topics, five research methods were covered by the projects (Table 3).

The SCM course implements the concept from Figure 1 as follows: method teams became *theory teams* for those students that did not want to carry out a study, but wanted to learn more details about a specific method or technique, e.g., the systematic review method [13]. Service teams became *cross-cutting teams* that build up a specific expertise and consulted theory and practice teams. The 20 teams were connected with each other, e.g., a theory team supported one or many practice teams, and both were supported by cross-cutting teams. The teacher supervised the individual teams as well as the groups of collaborating teams. The teams were formed right in the first session of the course and, thus, the projects became the main subjects to build the learning experience upon.

3.3.2 Integration in an Advanced Software Engineering Course

A course on *Advanced Software Engineering* (ASE, *University of Innsbruck*, *UI*, 2017) implemented the presented approach as part of a master-level course on software engineering in which empirical studies complemented the (technical) software engineering topics. These technical software engineering topics were organized around the concept of models in software engineering and covered software process models (including agile process models), modeling languages including UML and DSLs, model transformations as well as predictive models, e.g., for defect prediction. Figure 4 illustrates the overall organization of the ASE course showing the introduction parts and the active project parts.

The overall goal of this course was to teach students advanced topics in software engineering and to let students make the experience of the value that empirical studies have to support software engineering activities. Specifically, the major learning goals of the ASE course were defined as follows:

- After the course, students know and understand different advanced software engineering concepts.
- After the course, students know the basic empirical research methods and know how to utilize empirical studies in the different software engineering activities.
- After the course, students have shown their ability to practically apply one research method to a specific software engineering activity, conduct and report on a small research project.

In total, eight topics have been proposed to the students and students could apply for the topics. Finally,

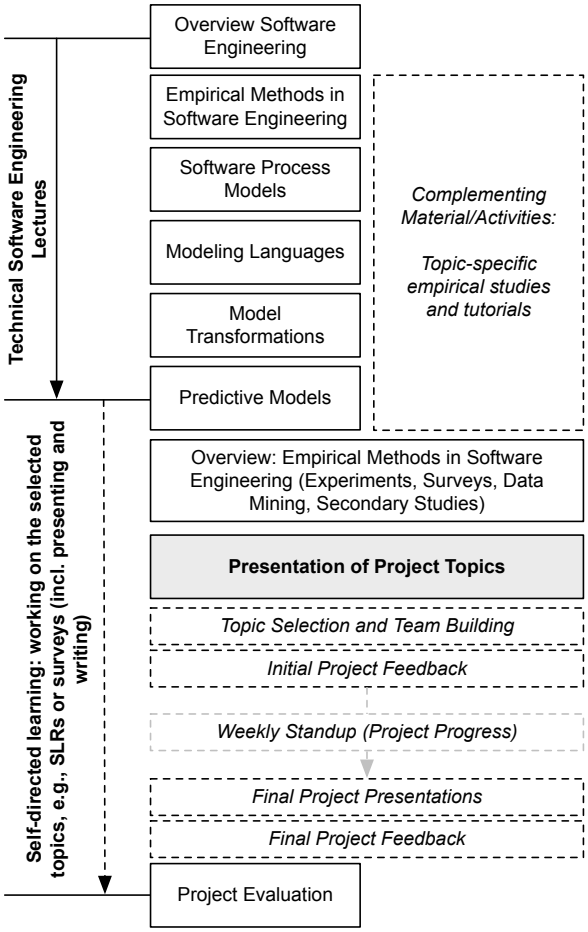

Figure 4: Overview of the topics and the general organization of the ASE course.

13 projects were started with two to three students each. The projects covered four research methods and six topics (Table 3).

The concept from Figure 1 was implemented as follows: the 13 project teams were formed as *practice teams*. Since the empirical studies were designed to complement selected topics of the ASE course, no explicit *method teams* or *service teams* have been formed. That is, all practice teams worked on specific topics and developed the technical skills and parts of the required methodological skills themselves. Additional methodological skills were delivered to the teams by the teachers and guest lectures, who also acted as advisors. Teams were formed when the mini-projects were assigned to the students.

4 Evaluation

In this section, we present the research design and evaluation strategy in Section 4.1, the results and a discussion in Section 4.2, and threats to validity in Section 4.3.

Research Questions	
RQ 1	*Do course-integrated empirical studies (mini-projects) help students improving their work approach?* We aim to study if course-integrated empirical studies (mini-projects) help students better understand the value of structured scientific work approaches. For this, we investigated the following detailed questions:
RQ 1.1	*Do mini-projects support a better/more effective learning?*
RQ 1.2	*Do mini-projects support a better understanding of concepts?*
RQ 1.3	*What are the perceived learnings of mini-projects?*
RQ 2	*Do course-integrated mini-projects help students better understand the role of empirical studies?* We aim to study the general attitude towards empirical studies, i.e., do students change their attitude once they actively conducted an empirical study. For this, we investigated two detailed questions:
RQ 2.1	*Do mini-projects change the attitude towards empirical studies?*
RQ 2.2	*Do course-integrated empirical studies studies help understanding challenges (revealing misconceptions)?*
RQ 3	*What are the perceived pros and cons of the mini-project approach?* We aim to study dis-/advantages perceived by students that participated in mini-projects.

Table 4: Overview of the research questions studied.

4.1 Research Design and Evaluation Strategy

We evaluated our approach in two master-level courses at two universities and by surveying the participating students. In this section, we present our research questions, outline the survey instrument, and describe the data collection strategies.

Research Questions To evaluate the proposed mini-project approach, we aim at answering the research questions listed in Table 4. Our three top-level research questions address the (general) learning experience, the usefulness of the approach in terms of improving the understanding of the role of empirical studies, and the perception concerning the pros and cons of the approach presented.

Data Collection To collect the data, we (initially) developed two online questionnaires for the SCM course based on Google Forms [14]. The first questionnaire was used in a mid-term evaluation; the second (extended) questionnaire was used for the final evaluation. While preparing the ASE course, we revised both questionnaires and conducted the first data collection before we started the mini-projects in the ASE course, and the second data collection, again, in the course's final evaluation when the mini-projects have been finished. All four questionnaires including a summary are available online[2].

Our questionnaire design allows for a 2-staged data collection that helps observing changing student perceptions and evaluating the courses over time. The questionnaires share a number of questions to allow for comparing courses, the implementations of our approach, and to qualitatively analyzing the student feedback. As a learning from the SCM data collection, the two ASE questionnaires put more emphasis on the single phases of the scientific workflow, e.g., by specifically asking for challenges and difficulties regarding the design of research instruments and conducting the data collection. Different to the questionnaires used in the SCM course, is the ASE questionnaires, students were asked to provide *nicknames* (to ensure anonymity), such that tracking individual students was possible to evaluate specific ratings and to evaluate such ratings over time.

Analysis Procedures All four questionnaires produce quantitative and qualitative data. For the quantitative analysis, we primarily use descriptive statistics to analyze the four measurements individually, over time per course, and for analyzing both courses. Furthermore, due to the questionnaire's evolution, for the ASE course we could conduct additional inferential statistical analyses, e.g., hypothesis testing and correlation analysis.

To qualitatively analyze the data, we used the free-text answers provided by the students. For the ASE course, an analysis of general learning and learning outcomes was performed using the questions for the expected learning outcomes, and the questions about the learning regarding the mini-project topics and the way of conducting empirical research (Appendix; variables MP_6–MP_8). An overall analysis of the course as such (in both courses) was performed using the questions for the courses' appropriateness, the lectures and exercises, and the perceived relation to practice (Appendix; variables GC_2–GC_4; interpreted as school grades). The coding of the feedback into categories was jointly performed by the two authors.

Validity Procedures To mitigate threats and to ensure the validity of the instrument, we reused a questionnaire design that was already applied to other courses and that received an external quality assurance [15, 18]. The original questionnaire design was extended by specific questions to evaluate the suitability of the mini-project approach.

Demographics In total, 68 students were enrolled in the SCM course of which 39 students participated

[2]Appendix: https://kuhrmann.files.wordpress.com/2018/11/appendix-draft.pdf

Category	SCM, n=38		ASE, n=29	
	Mean	SD	Mean	SD
Course complexity	2.87	0.66	2.62	0.55
Course speed	3.05	0.76	2.83	0.38
Course volume	2.58	0.91	2.10	0.76
Relation to Practice	2.11	0.99	2.34	1.03

Table 5: Results of the final course evaluation.

in the initial evaluation, and 38 in the final evaluation. In the ASE course, 33 students were enrolled, and 29 students participated in both evaluations. From the 29 ASE-students, 27 provided a *nickname* that was used in the subject-based analyses. Table 5 gives an overview of the general course evaluation. Students of both courses perceived the course complexity, speed and volume as moderate and see a good relation of the course to practice.

4.2 Results and Discussion

This section presents the findings of the evaluation of our proposed *course-integrated mini-project* approach. The following sections present the findings according to the research questions described in Table 4.

4.2.1 RQ 1: Improved Work Approach

With RQ1, we aim to study if course-integrated mini-projects help students understand the *value* of a structured scientific work approach. To better structure the findings, we defined three sub-questions (Table 4), which we discuss in the following.

RQ 1.1: Support for a better learning This sub-question is addressed by the answers to the statement: *"The mini-projects improve the learning experience"*, which was quantitatively analyzed.

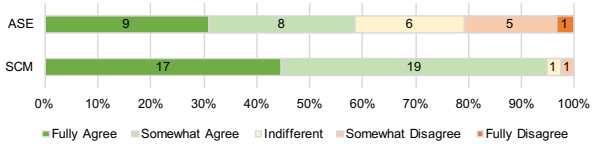

Figure 5: Results for the learning experience from final questionnaires (SCM: n=38, ASE: n=29).

Figure 5 shows the results (taken from the final evaluation) for both courses and shows that 95% of the SCM-students consider the mini-project approach contributing to an improved perceived learning experience (3% each rate the teaching format neutral or less effective than other teaching formats). For the ASE course, 59% consider the mini-project approach more effective, and 21% each rate it neutral or less effective. In summary, mini-projects contribute to an improved perceived learning experience, especially in the SCM setting, but also in ASE, where mini-projects were only one part of the course.

RQ 1.2: Support for a better understanding of concepts A key to provide value to the students is to make software engineering concepts better/easier to understand. For this, students were asked to rate the statement: *"The mini-projects helped me understanding concepts better"*, i.e., whether or not the understanding of concepts of interest has been improved. In this context, an investigation of the role of empirical studies is provided in Sect. 4.2.2. Again, the majority of the students (92% for the SCM course and 69% for the ASE course; Figure 6) considers the mini-project approach advantageous for gaining a better understanding of software engineering concepts.

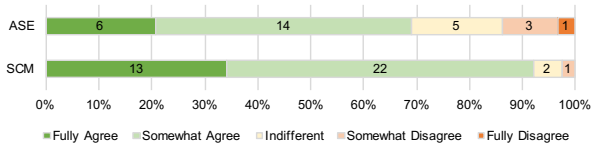

Figure 6: Results for the improved understanding from final questionnaires (SCM: n=38, ASE: n=29).

RQ 1.3: Perceived Learning The question for the perceived learning is answered using five statements (Appendix, $MP_{2.4}$–$MP_{2.8}$). In particular, we were interested to learn about the perceived impact to the later career (*"The practiced scientific work approach will help me in my later career."*) and shareable expertise built in the course (*"I built a specific exercise that I could share with other teams."*), and a retrospective rating of the group work in the mini-projects (looking back: *"contributed to my learning experience"*, *"team work [..] was good"* and *"collaboration [..] was good"*).

Figure 7 shows the aggregated results for the perceived learnings. Approximately 63% of the SCM students and 59% of the ASE students think that the courses provide take-aways that will have a positive impact on their later careers. Concerning the shareable expertise, 53% of the SCM students state that they have obtained knowledge and expertise that they can share with others; 29% are indifferent. In the ASE course, even though the course has more "practical" elements, only 41% of the students think that they built a shareable expertise, but 48% are indifferent.

Concerning the general perceived learning experience and the teamwork within the project team, the vast majority of the students rate the courses as good and very good. However, the cross-team collaboration shows a different picture—notably in the SCM course in which interdisciplinary work was enforced by the course design. In the SCM course, 29% of the students considered the cross-team collaboration good to very good, but 55% rated the cross-team collaboration bad to very bad. Analyzing this phenomenon, we found the necessity for the different teams to interact with each other to obtain required knowledge from other

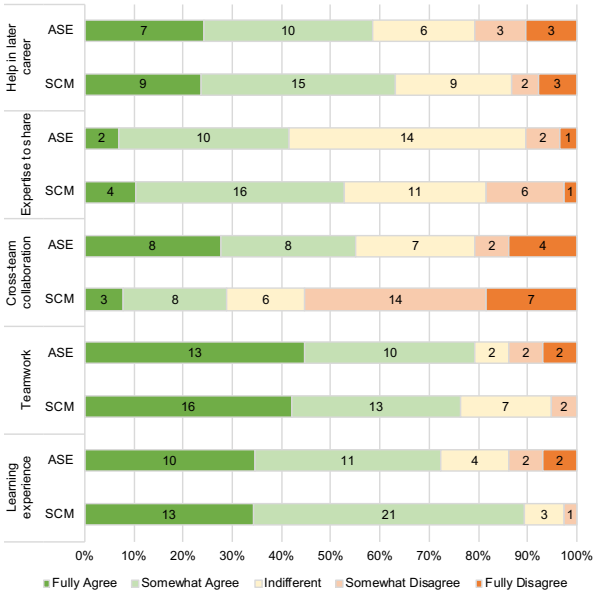

Figure 7: Results for the perceived leanings from final questionnaires (SCM: n=38, ASE: n=29).

teams by also trying to keep their own schedule the most disappointing aspect. On the other hand, we found an "understanding" for this kind of work, which reflects reality in interdisciplinary collaboration and, thus, students eventually considered this a significant learning. Considering the ASE course, we wanted to learn whether a similar behavior can be observed. As Figure 7 shows, cross-team collaboration is still considered a problem, even though the heterogeneity of the project teams and thus the need to collaborate was reduced.

An in-depth analysis of the perceived learnings of mini-projects was performed by qualitatively coding the responses of the free-form text questions (GC_1: *"What was your major take-home asset [..]?"*, MP_7: *"What did you learn about the topic of your research project?"* and MP_8: *"What did you learn about empirical research?"*). For GC_1, 26 students from the SCM course provided feedback. In the ASE course, 27 students provided feedback for MP_7 and MP_8. In total, we extracted 38 statements from the SCM course and 60 statements from the ASE course (for both questions). The students' statements were categorized based on keywords, and the threshold for building a category was set to three references.

Table 6 provides the condensed qualitative feedback on the perceived learnings in eight categories. Summarized, topic specific learnings (e.g., application of DSLs or comments in programming languages) as well as learnings related to the application of empirical methods (e.g., formulation of research questions or application of specific empirical methods like surveys) were frequently mentioned. Also, data management (i.e., data collection, preparation and analysis

Category	Total	SCM	ASE
Topic specific (i.e., mini-project topics)	16	2	14
Empirical methods (e.g. experiments)	16	6	10
Reporting findings (from studies)	13	13	0
Importance, meaning (emp. research)	12	2	10
Data management (in studies)	11	1	10
Effort (to plan/conduct a study)	10	0	10
Technical skills	3	2	1
Soft skills	17	12	5

Table 6: Qualitative feedback for the perceived learnings of mini-projects from final questionnaires.

of data) and reporting results of empirical studies are highlighted. Students experienced that conducting an empirical study causes effort and might be a complex endeavor (*"It is hard to get results, which have a strong meaning"*). On the other hand, students also built an understanding of the importance and the meaning of empirical research in software engineering (*"The topic exists and is very useful if done correctly"*). Finally, students hardly report learnings regarding technical skills (e.g., using LaTeX or R). However, manifold learnings about soft skills (e.g., reviewing techniques) are reported, notably concerning teamwork and cross-team collaboration (see also Figure 7).

4.2.2 RQ 2: Improved Understanding

This research question aims at investigating if students built an understanding of the role of empirical studies. Specifically, if students consider empirical studies a valuable instrument to complement the technical software engineering activities in a beneficial way.

RQ 2.1: Changed Attitude towards Empirical Studies To learn about the students' understanding of the value of empirical studies, we asked the students whether their view on empirical studies has changed once they actively conducted an empirical study themselves (*"I like the mini-project part"*).

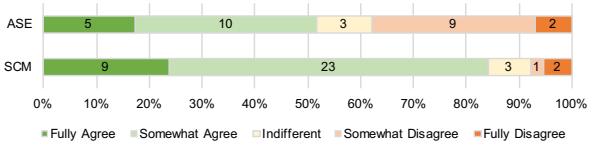

Figure 8: Results for the changed view on science from final questionnaires (SCM: n=38, ASE: n=29).

Figure 8 shows that 84% of the SCM students changed their view on science and the value of empirical studies after conducting an empirical study themselves. The ASE course provides a different picture. Still 52% of the students changed their view, but almost 38% did not change their view on science and empirical studies.

RQ 2.2: Challenges of Empirical Studies Besides the general perception of science and empirical studies, we are also interested in specific challenges. For this, we revised the questionnaire (see Appendix) and added questions that explicitly address the scientific work process.

Activity	Results	p-value
Definition of research questions	V = 80	0.5257
Working with scientific literature	V = 55.5	0.4927
Design of research instruments	V = 68	0.3254
Implementation of instrument	V = 43	0.7808
Data collection	V = 135	0.0277
Performing data analysis	V = 107	0.9529
Writing the report	V = 106	0.3698

Table 7: Results of the paired Wilcoxon signed-rank test for the ASE course (n=27).

To study the perceived challenges students face when implementing empirical studies, we asked the students to rate the different parts of the scientific work process (Appendix, variables $MP_{5.1}$–$MP_{5.7}$; see also Table 7). Furthermore, 27 students enrolled in the ASE course provided a *nickname*, which we used to investigate challenges over time by evaluating the initial and the final questionnaires. We performed a Wilcoxon signed-rank test to compare if there are significant differences between the initial perception of the scientific work process and the final one after the study has been performed. Table 7 shows the test results for the various parts of the work process.

The results show that only for the activity *data collection* there is a significant difference ($p < 0.05$). This indicates the perceived difficulties and challenges regarding the data collection changed for the participating students. A Spearman rank correlation coefficient for the initial and final feedback on the level of challenges regarding the data collection is low ($\rho = 0.2775$), which further indicates that there is only a weak positive correlation between between the data collection challenges perceived initially and the challenges perceived at the end. Figure 9 shows the uncorrelated perceived challenges regarding the different activities in the scientific work process (collected in the ASE course; initial and final questionnaire).

We conclude that the data collection is an activity in empirical studies for which challenges can be easily over- and underestimated. When teaching empirical software engineering it is therefore important to put special emphasis on the important role of data collection and its challenges to prevent students from over- or underestimating the required effort.

4.2.3 RQ 3: Perceived Dis-/Advantages

The third research question aims to investigate the perceived advantages ("pros") and disadvantages ("cons") of the mini-project approach. For this, we qualitatively analyzed the students' feedback by coding the responses of the free-form text questions (GC_2: *"Up to 5 things that were good"* and GC_3: *"Up to 5 things that were bad"*).

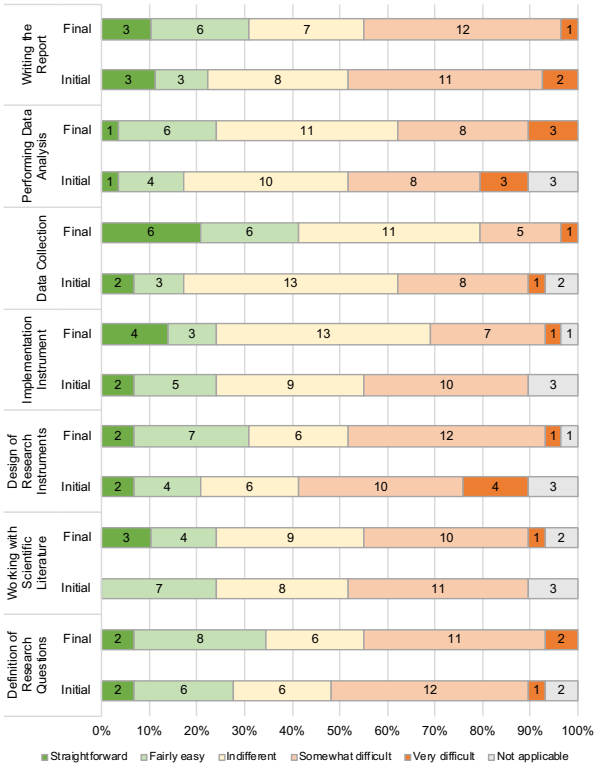

Figure 9: Overview of the perceived challenges on the different scientific work activities in the ASE initial and final questionnaire (n=29).

In the SCM course data for GC_2 and GC_3 was collected in the initial and final questionnaire. For the ASE course, data was collected in the final evaluation only. Hence, for the data analysis presented in the paper at hand, we only consider the data collected in both final evaluations. In the SCM course, 29 students provided comments in the final evaluation. Respectively, 21 ASE students provided feedback on perceived dis-/advantages. In total, we extracted 72 *pro*- and 42 *con*-statements from the SCM feedback, and we extracted 54 *pro*- and 23 *con*-statements from the ASE feedback. Both feedback sets were categorized and analyzed based on keywords (qualitative coding), whereas the threshold for a category was set to three mentions. Table 8 provides the aggregated qualitative feedback on the perceived *pros* and *cons* of the mini-project approach in nine categories grouped by SCM, ASE, and in total.

General Perception In summary, the mini-project approach was seen very positive. For both courses SCM and ASE, the students identified considerably more *pros* than *cons* on the mini-projects. In both settings, i.e., SCM (a self-contained empirical software engineering course) and ASE (a part of a software

Category	SCM		ASE		Total	
	👍	👎	👍	👎	👍	👎
Structure, organisation	18	14	9	11	27	25
Knowledge transfer	18	3	3	2	21	5
Mini-projects	14	3	6	2	20	5
Research, tech. skills	6	2	8	1	14	3
Topics	6	4	7	0	13	4
Relevance	3	3	9	0	12	3
Guest lectures	5	7	6	0	11	7
Group work	2	1	6	1	8	2
Effort	0	5	0	6	0	11

Table 8: Qualitative feedback on the perceived pros and cons of mini-projects from final questionnaires.

engineering course), the obtained research-related and technical skills were perceived very positive. The topics covered in the courses were positively evaluated as well; especially in the ASE course in which mini-projects were integrated as a part of a software engineering course and focused on current (hot) topics in software engineering. Feedback and knowledge transfer were especially highlighted and considered positive in the SCM course with its different types of collaborating teams (dedicated practice, method and service teams), in-depth introductions to empirical methods, and introductions and exercises in scientific reading and writing. Also, group work was generally considered positive, yet, the ASE course with its more uniform teams was perceived more advantageous. As already discussed in Sect. 4.2.2, the setting from the SCM course suffers from the teams' heterogeneity and the necessity to establish a cross-team collaboration, which caused more effort for the project teams.

Guest Lectures In both courses, guest lectures were given by external researchers. Considering the outcomes from Table 8, guest lectures in the ASE course were considered positive, whereas the guest lectures in the SCM course received a more indifferent rating (with a slight tendency towards a negative evaluation). We argue that this perception is related to the selection of the speakers rather than the course setting, yet, this remains subject to further investigation.

Course Organization From the organizational perspective, the courses were perceived indifferent and a relatively high number of positive and negative comments was provided. On the positive side, students highlighted the organization and structure in general ("*Organization was good*"), and, in particular, also the course assessment mode ("*Fair grading system*") and the sequence of activities ("*The mini-projects were structured well—good planning of when to do the work, when to make a presentation and when to turn in the paper*"). On the negative side, the overall flow was mentioned ("*Things started to slow down way too much after the first 5 lectures*") as well as the speed of the lecture ("*A little slow lectures*") and the general scheduling and coordination of the lecture with other obligations ("*During examination time, time overlap with studying*").

Effort Finally, the effort caused by the courses was perceived negative in both settings. Feedback for the SCM course says *"The volume does not fit well a 5 ECTS point course"* and, respectively, for the ASE course *"Sometimes a single task was too big"*. This feedback reflects a side-effect of project work that typically causes more effort than closed tasks—or even "listen-only" classes. However, such comments motivate a revision of the projects, e.g., splitting tasks into smaller units to better support continuous work and to reduce the perceived effort in a short time frame, but still keep the learning effect of performing empirical research as we received it also from the SCM comments *"in my opinion learning the scientific method without working with the methods it's only knowing about the methods, not learning them."*

4.3 Threats to Validity

We discuss issues, which may have threatened the construct, internal and external validity as well as measures to mitigate them.

Construct Validity The construct validity might be threatened by the two different implementations of the approach presented and the instrument used for its evaluation. Although the two courses differed with respect to the applied integration strategy for mini-projects, for both courses, we used the same yet tailored questionnaire and combined the responses. To increase construct validity, we developed the questionnaire from an external source [15], which both authors reviewed and evolved. Furthermore, we collected and combined quantitative and qualitative data to answer and discuss the different research questions.

Due to the overall setup, the questionnaires differed between the courses. Hence, some analyses like the individual-based investigation of challenges over time (RQ 2.2) were possible in the ASE course only. Furthermore, analyses regarding perceived learnings of the students had to be performed using different questions (SCM: GC_1, ASE: MP_7 and MP_8; see Appendix), which was handled using a multi-staged coding process that resulted in common categories.

Internal Validity The internal validity might be threatened by the rather low number of participants and the participants' self-reporting, which both might affect the relationship between course-integrated mini-projects and the investigated effects on learning and understanding of concepts and the role of empirical studies in software engineering. To mitigate these threats, we studied the integration of mini-projects

in two settings with an acceptable number of participants (SCM: 39 and ASE: 29). We also introduced the questionnaire to the students to minimize the risk of misinterpretation.

To triangulate the results of quantitative analysis and to investigate the relationship between course-integrated mini-projects and their effects holistically, we also applied qualitative analysis to analyze the responses of the participating students.

External Validity The external validity might be threatened by the issue of the rather low number of settings in which the course was performed and evaluated. However, we implemented and evaluated the course for each of the two course integration strategies proposed in Section 3.3, i.e., self-contained empirical software engineering course and integration in software engineering course. Two researchers from two different institutions were involved in preparing, conducting and evaluating the courses. Furthermore, we combined quantitative and qualitative data to get a broader view on teaching empirical software engineering with course-integrated mini-projects.

The participants' self-reporting might also affect the generalizability of the results. To mitigate this threat, we introduced the questionnaire to the students to minimize the risk of misinterpretation. Since the paper at hand is an initial study, we consider further in-depth analyses, e.g., of course artifacts like final reports, and replications of the courses as well as the transfer of the approach presented and its evaluation as future work.

5 Conclusion

In this paper we presented and evaluated an approach to teach empirical software engineering with course-integrated mini-projects. In such mini-projects students conduct small empirical studies in collaborating teams within a software engineering course or a research methods course. We illustrated the approach by presenting two integration strategies: first a self-contained course on empirical software engineering given at the *University of Southern Denmark* and second as part of an advanced software engineering course given at the *University of Innsbruck*. Both implementations were complemented by a study that showed a positive learning experience and an improved understanding of (empirical) software engineering concepts.

More than half of the participating students state as a perceived learning of the course that empirical studies are helpful for their later careers (systematic and evidence-driven work). In addition, a statistical analysis revealed that especially the data collection activities are underestimated by the students, which allows for future improvements of university courses. We consider this aspect critical not only for empirical software engineering in particular, but also due to the increased importance of *Data Science*, *Machine Learning* and *Artificial Intelligence* courses or even programs in general. In all these setting, effective and efficient data collection and preparation for further analysis is essential. Hence, we encourage other teachers to put special emphasis on all data-related activities, not only the data analysis.

In summary, students provided an overall positive qualitative feedback on the course-integrated mini-projects as well as on the skills achieved and their relevance. This motivates to further explore and disseminate the presented approach. However, on the downside, the students pointed out the overall flow of the courses and the perceived high effort to perform mini-projects, which requires a further refinement of the courses, e.g., by splitting it into smaller tasks of uniform granularity. In future, we will therefore revise our approach accordingly and further disseminate it. We also plan to perform further in-depth analyses of course artifacts, e.g., the students' final reports, as well as replications of the courses.

Finally, this paper presents an approach that has been implemented twice and it also provides initial data. To get further insights and to improve the data available, we cordially invite other teachers to adapt and integrate our approach into their courses and to share their experiences.

References

[1] L. W. Anderson and D. R. Krathwohl, editors. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged Edition*. Pearson, 1st edition, 2000.

[2] V. Basili, R. Selby, and D. Hutchens. Experimentation in software engineering. *Trans. on Software Engineering*, 12(7):733–743, 1986.

[3] E. Dale. *Audiovisual methods in teaching*. Dryden Press, 3 edition, 1969.

[4] J. Dillon. A Review of the Research on Practical Work in School Science. Technical report, King's College, 2008.

[5] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch. Building blocks for continuous experimentation. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, RCoSE 2014, pages 26–35, New York, NY, USA, 2014. ACM.

[6] F. Fagerholm, M. Kuhrmann, and J. Münch. Guidelines for using empirical studies in software engineering education. *PeerJ Computer Science*, 3(e131), September 2017.

[7] D. Fucci and B. Turhan. A replicated experiment on the effectiveness of test-first development. In

International Symposium on Empirical Software Engineering and Measurement, pages 103–112. IEEE, Oct 2013.

[8] D. Fucci, B. Turhan, and M. Oivo. Impact of process conformance on the effects of test-driven development. In *International Symposium on Empirical Software Engineering and Measurement*, pages 10:1–10:10. ACM, 2014.

[9] V. Garousi, M. Felderer, M. Kuhrmann, and K. Herkiloğlu. What industry wants from academia in software testing?: Hearing practitioners' opinions. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, pages 65–69, New York, NY, USA, 2017. ACM.

[10] V. Garousi, M. Felderer, and M. V. Mäntylä. The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, page 26. ACM, 2016.

[11] V. Garousi, M. Felderer, and M. V. Mäntylä. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 2018.

[12] M. Kasunic. Designing an effective survey. Technical report, Carnegie Mellon University Pittsburgh Software Engineering Institute, 2005.

[13] B. A. Kitchenham, D. Budgen, and P. Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press, 2015.

[14] M. Kuhrmann. Teaching empirical software engineering using expert teams. In *SEUH*, pages 20–31, 2017.

[15] M. Kuhrmann, D. M. Fernández, and J. Münch. Teaching software process modeling. In *International Conference on Software Engineering*, pages 1138–1147, 2013.

[16] M. Kuhrmann, D. Mendez Fernández, and M. Daneva. On the pragmatic design of literature studies in software engineering: an experience-based guideline. *Empirical Software Engineering*, 22(6):2852–2891, Dec 2017.

[17] 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)*. IEEE, 2016.

[18] 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*, ICSE, pages 412–421. ACM, May 2016.

[19] M. Kuhrmann and J. Münch. Enhancing software engineering education through experimentation: An experience report. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–9, June 2018.

[20] K. Labunets, A. Janes, M. Felderer, and F. Massacci. Teaching predictive modeling to junior software engineers—seminar format and its evaluation: poster. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 339–340. IEEE Press, 2017.

[21] J. Linåker, S. M. Sulaman, R. M. de Mello, and M. Höst. Guidelines for conducting surveys in software engineering. Technical report, Lund University, January 2015.

[22] J. Parker. *Using laboratory experiments to teach introductory economics*. Working paper, Reed College, http://academic.reed.edu/economics/parker/ExpBook95.pdf, accessed 2014-10-23.

[23] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattson. Systematic mapping studies in software engineering. In *International Conference on Evaluation and Assessment in Software Engineering*, pages 68–77. ACM, 2008.

[24] K. Petersen, S. Vakkalanka, and L. Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Inf. Softw. Technol.*, 64:1–18, August 2015.

[25] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.

[26] 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*, volume 4336 of *LNCS*, pages 135–142. Springer, 2007.

[27] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[28] H. Zhang, M. A. Babar, and P. Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53(6):625–637, 2011.