Educating Future Software Architects in the Art and Science of Analysing Software Data

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens AG, Corporate Research, Munich
Germany
wolfgang.mauerer@oth-regensburg.de

Stefanie Scherzinger
Technical University of Applied Sciences Regensburg
Regensburg, Germany
stefanie.scherzinger@oth-regensburg.de

***Abstract*—We report the design and teaching experience of a Master-level seminar course on quantitative and empirical software engineering. The course combines elements of traditional literature seminars with active learning by scientific project work, in particular quantitative mixed-method analyses of open source systems. It also provides short introductions and refreshers to data mining and statistical analysis, and discusses the nature and practice of scientific knowledge inference. Student presentations of published research, augmented by summary reports, bridge to standard seminars. We discuss our educational goals and the course structure derived from them. We review research questions addressed by students in mini research reports, and analyse them as tokens on how junior-level software engineers perceive the potential of empirical software engineering research. We assess challenges faced, and discuss possible solutions.**

***Index Terms*—Empirical Software Engineering, Teaching Quantitative Methods, Statistical Analysis, Literature Seminar**

I. Introduction

Effective decision making is a crucial part of being successful in software engineering (SWE). Architects, programmers and even technical managers need to decide, among others, how to best organise team collaboration, how to choose appropriate software components and frameworks, and how to design entire software architectures.

Scientifically sound decision making is ideally based on measurable facts. Consequently, substantial portions of SWE research rest on empirical, quantitative methods. This constitutes a teaching challenge: Beyond covering an already large syllabus, advanced statistical methodology must be introduced, to create an understanding of the benefits and limits of scientific knowledge inference.

In the Master-level seminar described in this paper, we address these challenges in a setting targeted at advanced students with a focus on practical engineering: We augment a traditional scientific seminar—avoiding to impose undue workload—with active, creative learning components, challenging students with the quantitative, data-driven investigation of a research question of their choice. At the same time, we re-use this setting to learn from our students (many of who are part-time employees in the local software industry,[1] or have previous work experience[2]) how empirical methods are observed by junior-level software professionals.

[1] In-house surveys show that 40% of all Master students dedicate over 40% of their time to casual work (usually as programmers). Details upon request.

II. Course Design

The computer science Master curriculum at *Technical University of Applied Sciences Regensburg* requires students to complete a scientific seminar worth 5 ECTS credits. In the following, we detail organisation, learning goals and timeline of the course. So far, we have taught two iterations.

A. Learning Goals

The course description for the scientific seminar[3] states these learning goals: The students learn to 1) independently research an area within the field of computer science, 2) critically reflect and summarise central ideas of scientific work, 3) perform literature search and reviews, 4) give a professional presentation, and 5) engage in an academic discussion.

To reach these goals, scientific seminars traditionally comprise a *seminar presentation* as well as a *seminar report* on an existing body of research. However, we also made it our goal that students actively experience empirical SWE (eSWE), beyond merely analysing existing research. They should gather background knowledge as to why (and when) an empirical, quantitative approach is preferable over more orthodox SWE, and experience benefits, limitations and challenges of quantitative work. Consequently, we desire that they 1) do not merely read up on principles, but acquire a certain level of proficiency in using and also mining version control systems[4], 2) gain first-hand experience with the technical and conceptual pitfalls in exploring a research question, 3) write a mini research report as a "training" opportunity before handing in the graded seminar report, and 4) are aware that not only technical aspects of building software, but also socio-technical and social aspects of software development can be quantified.

[2] All undergraduate students at *Technical University of Applied Sciences Regensburg* complete a mandatory, 18-week internship. Additionally, 40% of Bachelor graduates report in in-house surveys that they have held full-time occupations in the private sector before taking up their studies.

[3] This course is detailed in the department module guidelines.

[4] The ubiquitous version control system `git` is an obvious choice, since it is a popular data source in research; using the system for data engineering usually implies a proficiency boosts in daily work, too.

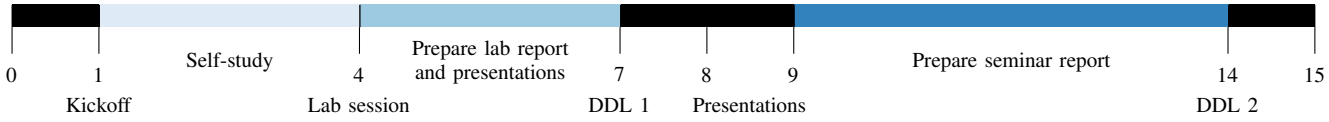

Fig. 1. Timeline breakdown of the 15 week term. "DDL" denotes a student deadline for submitting or presenting results.

B. Organisation and Timeline

Figure 1 summarises the timeline of our seminar, broken down across the 15 week term, and highlights the main events.

a) Kickoff session (week 1): Students enrolled in the Masters program are assigned to one of several parallel seminar tracks (organised by different professors), according to their topical preferences. A track comprises 20 participants.

We asked our students to prepare the online course "Version Control with Git".[5] This course includes hands-on exercises, so our course participants can operate `git` directly on the command-line (and not just via feature constrained colourful user interfaces). This includes advanced working with different branches, cloning, fetching, forking, and cherry picking, as well as a basic understanding of the data storage model.

b) Lab Session (week 4): The lab session is an all-day workshop where the students focus on practical exercises. This includes answering questions on more advanced aspects of `git` (the full list is available in the online supplement.[6] This allows students to self-assess their level of proficiency in handling `git`. (Additionally, we schedule two papers [1], [2] on the subject early in the paper presentation stage, see *(d)*).

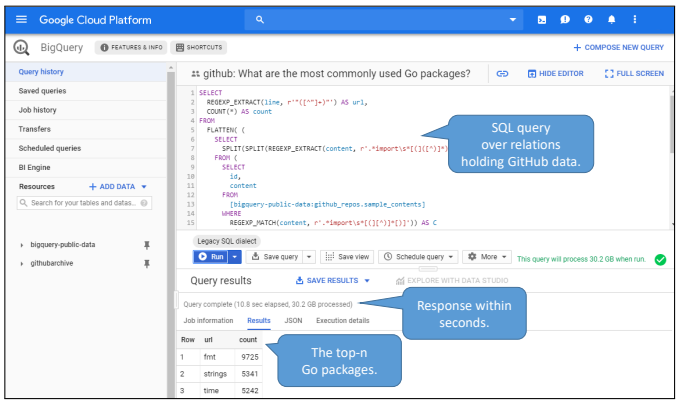

Fig. 2. Using BigQuery to identify the most frequently imported Go packages.

Likewise, we provide small challenges that must be solved using Google BigQuery. BigQuery is a cloud-based data warehouse.[7] It provides various open data sets, among them the GitHub activity data. As of October 2019, this contains a snapshot of open source software (OSS) repositories amounting to over 3 TiB of data (currently, over 2.8 million repositories, 145 million unique commits, and over 2 billion different files). Queries such as "What are the most frequently imported Go packages?" may be stated declaratively, using SQL, as Figure 2 illustrates. Together, we discussed the issue of reproducibility, as the data collection is updated regularly.

We further provided a refresher on statistics and on scientific writing, the latter based on Zobel's book [3].

For a mandatory two-page mini research report (discussed in detail in Section III-B) that explores a self-chosen research question, students collected ideas in an interactive brainstorming session,. We gave feedback on the validity and feasibility of each question, taking into account the temporal constraints. We also commented on apparent threats to validity.

c) Mini research report deadline (week 7, Deadline 1): We graded the submitted mini research reports by detailed criteria that we made public beforehand[8].

d) Seminar presentations (weeks 8 and 9): Each student is assigned one (usually seminal) original research paper, or a book chapter from Ref. [4], as a basis for the seminar presentation and report. Prior to presentation and discussion, students were mentored one-on-one, like in traditional seminars.

e) Seminar report (week 14, Deadline 2): The five page seminar report is prepared by week 14. It wraps up the core ideas of the underlying article or book chapter, and discusses it critically in the context of related work, methodological soundness[9], and practical utility.

III. Experience Report

We next report on our experience. We begin by discussing Google BigQuery as a means of evaluating research questions. We then reflect on the mini research reports. We review encountered challenges in the upcoming section.

A. Data Provisioning with git and Google BigQuery

Felderer and Kuhrmann [5] confirm that students tend to underestimate the effort of data collection and preparation, in agreement with common experience in data science. This calls for using sophisticated tools that come with powerful data preparation pipelines. Yet unfortunately, we found that many of the software solutions used by professional researchers lack in quality and maturity, particularly regarding ease of installation and setup, completeness of documentation, and usability, which was confirmed after consultation with the tool authors. In short, we failed to get any of the state-of-the-art tools[10] used in academic research to work for in-classroom

[5]The course is available on the Udacity MOOC platform.

[6]Blue coloured text provides a link in the electronic version of this paper.

[7]While BigQuery is a commercial service, it can be used without billing enabled, but requires that students are comfortable with a Google account.

[8]The grading rubric is available in the online supplement.

[9]This requires substantial individual guidance from the instructors. Additionally, the statistical refresher points out commonly encountered problems.

[10]Easy to install and use tools like `gitstats` are too simplistic even for less ambitious research questions chosen by students.

Students' Research Questions

- 30%—Relationships between straight-forward observables
  Time of day versus bug introduction?
  Does the number of bugs per developer vary with project age?
- 23%—Velocity of changes to observable quantities
  Speed of Java dependency updates after the weekly security issue?
  How fast are bug tickets closed?
- 13%—Testing: effort, coverage, and utility
  How are unit tests distributed by programming language?
  How does test coverage evolve?
- 10%—Hidden and indirect project properties
  How many OSS projects are company supported?
- 10%—Test (anecdotal or established) SWE conjectures
  Developer group size versus the 7±2 scrum assumption?
  Do code of conducts have measurable effects?
- 10%—Trivia — *Do bigger files change more often?*

Fig. 3. Distribution of student research ideas, categorised (subjectively) by topic, along with typical research questions.

use within reasonable effort (we grudgingly need to accept a share of the blame since this also holds for our own tools).

Thus, research questions based on complex socio-technical observations or multi-modal data sources cannot be addressed in mini research reports. To compensate, we devote a substantial share of the discussed literature on such research (*e.g.*, the seminal series of papers on socio-technical congruence by Cataldo, Herbsleb and co-workers, initiated in Ref. [6]).

We settled on two recommendations for how students can conduct their own research. (1) First, we proposed individual, programmatic analysis using either scripted calls of `git` or (preferably) using `git` front-end libraries from scripting languages for data collection (we recommend PyGit2, GitPython, and Git2R). (2) Alternatively, we proposed to use BigQuery, as already discussed. The well-curated data relations of the latter alleviate common issues that trouble the collection of "big" data – students can focus on writing SQL.

Following these recommendations can reduce the effort spent with data ingestion nuisances like parsing (broken) dates, parsing (broken) strings, handling (broken and/or mixed) encodings, or handling other (broken) system details.

B. Mini Research Reports

Mini research reports could be produced by teams of two, and students had free choice on the topic. Each run of the seminar produced about 20 suggestions with some overlap, resulting in 30 unique candidate questions (the full list of candidate questions is available in the online supplement). We identified six topical groups, as shown in Figure 3, along with typical research questions. We also show the distribution of the questions according to our categorisation.

We additionally categorised each research question concerning the research methodology: 1) Scope: Is the research question related to a single project or does it pertain multiple projects? 2) Analysis Method: Is a simple (count-based) measurement considered, are (correlations or stronger forms of) relationships *between* measured variables addressed, or does the research question try to resolve a specific hypothesis? 3) Time Resolution: Is the question applied in a time-resolved way (*i.e.,* did students consider that properties may change over time), or is each project analysed as single *static* entity?

Figure 4 provides a mosaic plot [7] of the resulting three-way contingency table. The largest group concerns the analysis of several projects, and considers relationships between variables—but without accounting for possible changes in the relationship over time. At the same time, no explicit testing of a hypothesis on a single project was suggested.

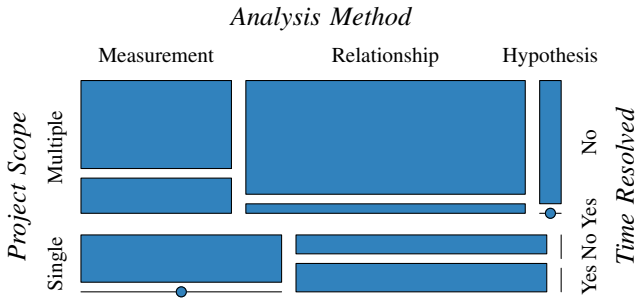

Fig. 4. Classification of mini research questions by methodological properties. Areas are proportional to the occurrence count of combinations, and flat lines denote absent combinations.

Discussion: We believe that to some degree, the topics chosen for mini research reports mirror the students' expectations and way of thinking: All seminar participants hold a Bachelor degree. Thus, they are fully qualified as junior-level developers. A survey after the winter 19/20 run showed that 75% of the participants have substantial work experience – 50% claimed work experience equivalent to about one year of full time employment, 25% even more than three years; details online. The students' intuition should therefore reflect on the intuition about eSWE in practice.

Most research questions proposed by the students concern measuring multiple projects instead of in-depth evaluations of a single one. Interestingly, *all* mini research reports involve quantitative measurements, and do not suggest any ethnographic or qualitative research, which does not mirror the topical distribution observed for published work. More than half of the research questions concern *relationships* between observed variables. This might indicate that students are interested in finding universal relationships valid beyond the scope of one particular undertaking, which meets our expectations towards Master-level students.

Usually, either a visual description or simpler measures like correlations or a univariate linear regression model are employed. Given the short time frame, this is understandable, but it might also indicate unease with more advanced analysis techniques. No team chose a machine learning-type analysis, despite the popularity of these methods among students.

Straightforward measurements of a *single* variable are usually intended to act as proxy for a (explicitly given, but often only diffusely defined) quality property. For instance, the number of tests is used as proxy for code quality, and number

and staleness of `TODO` entries in the code proxies for project progress. Students did not consistently realise that relations between proxy and indirect observables are not always in direct proportion, and that assuming such connections in the first place is a threat to validity. Thus, some research questions might even be categorised as "bad smells", as defined by Menzies and Shepperd [8]. Of course, we do not hold this against our students, who are novices in eSWE. Rather, we hope that by attending the seminar, the students learn to recognise "smelly" research questions.

Interestingly, hardly any students set out to apply principles and measures that are part of the standard SWE curriculum [9] of their SWE lecture, such as code metrics or code coverage.

Overall, we typically see simple statistical analyses for relations. What is missing is the question on how any of the measured co-variables influence quality or other properties of projects, or can even induce actionable consequences. This indicates that prior to the seminar, there was no established notion if and how complex decisions in SWE projects can be based on evidence- and measurement-based reasoning. Only 50% of the participants of the winter 19/20 run reported prior *literature* experience with eSWE methods; interestingly, no one reported prior use of eSWE in commercial projects.

IV. Challenges

In the following, we highlight several challenges that we encountered in teaching the course.

A. Scientific Method: Theory and Application

In the computer science curriculum at *Technical University of Applied Sciences Regensburg*, the scientific seminar is only taught at the Master level. This exposes students later than desirable[11] to scientific processes and methodology, and to conducting systematic research.[12] Students usually need to sharpen their understanding on the differences between hypotheses, theories, laws, observations, and conjectures, that is, the basic building blocks of scientific insight, as we frequently observe when supervising student theses.

Both authors have worked in industry, and have professionally built commercial software, before returning to academia. We find that exposure to the scientific method is useful for properly evaluating and understanding contemporary results of empirical software engineering research (Q1), for assessing the value of marketing claims of commercial vendors (Q2), and for comparing the novelty of approaches and solutions to the large body of existing work (Q3). The survey results in Figure 5 show that students mostly agree, except for Q2. The students share our enthusiasm that eSWE methods will help them become better software engineers (Q4), although confusingly only one student in four plans to employ such methods in the future (Q5).

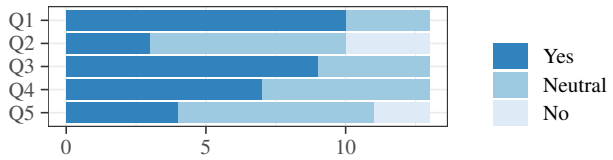

Fig. 5. Student opinion after the winter 19/20 run regarding the usefulness of eSWE and scientific methods in practice (questions Q*n*: see text).

The attitude towards philosophical aspects of science versus the acquisition of practical knowledge is, for many students, not unambiguously in favour of the former. Two aspects require particular attention in teaching: Firstly, software engineering comprises technical *and* social aspects, and it is usually impossible to derive quantitative a-priori theories in fields with such characteristics. Statistical inference therefore needs to be understood as the predominant means of establishing certainty. Many statements that prevail in the industrial domain—however credible they may sound from "experience"—can only be rationalised or refuted in this way.[13] Secondly, conducting a too delicately faceted discussion on the nature of science would distract from the seminar core. Differences between scientific research and actions dictated by practical necessity can be exposed by entertaining the pragmatic viewpoint of equating scientific insight with systematicity [10].

Providing or refreshing the aforementioned knowledge necessitates covering a substantial body of topics that often exceed what is covered in the non-elective parts of the curriculum. The lab session contains general guidance on these issues, but we further equip students with a comprehensive slide deck that details some of the aspects, and contains appropriate pointers for self-study. Care is needed to not put undue burden (or any perception of undue load) on the participants, to keep the workload comparable between parallel seminar tracks.

B. Statistics, Machine Learning and Data Analysis

Software engineering research rests on a wide body of statistical methods, but is also sometimes known to employ these techniques in inappropriate or flawed ways [11]. We believe this implies three challenges that need to be solved:

Firstly, popular statistical methods in research (such as advanced forms of multivariate regression, mixed models, association rules etc.) are usually not covered in compulsory undergraduate lectures. Secondly, students found it challenging to apply their method knowledge to practical data sets (*e.g.*, knowing the principles of linear regression is not sufficient to interpret the comprehensive output delivered by statistical software, as is evaluating quality or aptitude of models for a

[11]Experiences from multiple half-day refresher courses on scientific data evaluation for early-stage PhD candidates confirmed, as far as the value of anecdotal evidence goes, that opportunities for improvement are not exclusively restricted to early-stage Master students.

[12]Related lectures include a compulsory course on Automata, Formal Languages and Computation (4 ECTS) that discusses nature and limits of scientific inference; a checklist for preparing a scientific experience report on a mandatory industrial internship; the preparation of a Bachelor's thesis (12 ECTS, albeit often performed in industrial settings); and an elective short course on conducting research, intermittently taught by the authors of this paper. The omission of a dedicated course on scientific procedure is in line with the German computer science curriculum recommendations [9], and therefore probably extends to many other academic institutions as well.

[13]It seems not entirely impertinent to remark that many popular textbooks on the decades old agile credo do not ease the situation.

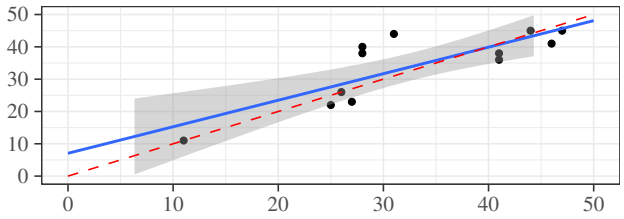

Fig. 6. Comparson of grading results (winter 18/19 run, 50 points maximum), together with a simple linear regression model (solid black), ideal correlation curve (dashed red), and 95% confidence interval (shade of grey).

given body of data). Thirdly, students predominantly perceive minimising the prediction error in statistical models as the sole quality criterion—most likely caused by the current surge of interest in machine learning and artificial intelligence—, which overshadows other schools of statistical thinking [12]. Software engineering research, in particular, is often concerned with parsimonious and interpretable models, and it was, for instance, necessary to remind students that common measures like the ubiquitously used R^2 value in linear regression are sub-optimal discriminators to judge models, since closeness of a model to data can (with over-fitting in mind) usually not immediately be related to model quality.

C. Availability of Full-Fledged Textbooks

We are not aware of a textbook for SWE that is not an edited collection of contributions by a large number of authors, or a collection of (essentially) research papers. We therefore decided to blend chapters from Ref. [4] with selected scientific works on research issues, in particular Refs. [1], [2], augmented by Easterbrooks *et al.* [13] on method selection for empirical research. Especially for presentations that establish base method knowledge, students identified differences in technical depth, scientific rigour, and focus, perhaps not entirely unjustified. Fully escaping this problem in a setting that discusses original research seems impossible.

D. Grading Based on Methods Preached

The difficulty of grading SWE projects is well known [14], and extends to student work produced in this seminar. Our major learning objective is to create awareness for data-driven methods, so we found it pertinent to hold grading to this standard. As an experiment, the mini research reports were therefore independently graded by both authors, and results were subjected to various statistical analyses and comparisons, which showcases their practical utility on an issue exposed to much student curiosity. Fig. 6 does not only demonstrate a satisfactory consistency between graders, but can also be used to remind students on the implications of residual correlation.

V. Related Work and Conclusion

The idea of students writing mini research reports has been pursued before. Our concept of mini research reports best matches the *experiments* proposed by Fagerholm *et al.* [15] and Ref. [16], the former of which gives detailed guidelines for including empirical studies in SWE education.

Researchers suggested ideas how to enable students to build up skills in eSWE. Wohlin [17] proposes (i) integration with a software engineering course, (ii) a stand-alone course, or (iii) a dedicated research method course. Fagerholm *et al.* [15] suggest to use eSWE methods as Master's thesis topics, which creates person-specific in-depth understanding, but unfortunately does not widely distribute method awareness. Option (iii) best matches our scientific seminar, whereas [5], [18], [15] report on courses that match options (i) and (ii).

For other courses comprising mini eSWE projects, students reported hands-on experience as beneficial for their future careers [5], which confirms our motivation and matches our experience after two iterations in a high teaching-load, application-oriented environment.

References

[1] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *6th IEEE Int Working Conference on Mining Software Repositories*, 2009.

[2] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of the 11th working conference on mining software repositories*, 2014.

[3] J. Zobel, *Writing for Computer Science*, 3rd ed. Springer Publishing Company, Incorporated, 2015.

[4] C. Bird, T. Menzies, and T. Zimmermann, *The Art and Science of Analyzing Software Data*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2015.

[5] M. Felderer and M. Kuhrmann, "Using Mini-Projects to Teach Empirical Software Engineering," in *Tagungsband des 16. Workshops "Software Engineering im Unterricht der Hochschulen"*, 2019, pp. 75–86.

[6] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. of the 2nd ACM-IEEE International Symposium on eSWE and Measurement*, ser. ESEM '08. New York, NY, USA: ACM, 2008, pp. 2–11.

[7] K. Hornik, A. Zeileis, and D. Meyer, "The Strucplot Framework: Visualizing Multi-way Contingency Tables with vcd," *Journal of Statistical Software*, vol. 17, no. 3, pp. 1–48, 2006.

[8] T. Menzies and M. Shepperd, "Bad smells in software analytics papers," *Information and Software Technology*, vol. 112, pp. 35 – 47, 2019.

[9] Gesellschaft für Informatik, "GI-Empfehlungen Bachelor-Master," https://gi.de/fileadmin/GI/Hauptseite/Aktuelles/Meldungen/2016/GI-Empfehlungen_Bachelor-Master-Informatik2016.pdf, 2016, [Online; accessed 08-Jan-2020].

[10] P. Hoyningen-Huene, "Systematicity: The Nature of Science," *Philosophia*, vol. 36, pp. 167–180, 06 2008.

[11] R. P. Reyes, O. Dieste, E. R. Fonseca, and N. Juristo, "Statistical Errors in Software Engineering Experiments: A Preliminary Literature Review," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18, 2018, pp. 1195–1206.

[12] L. Breiman, "Statistical modeling: The two cultures (with comments and a rejoinder by the author)," *Statistical science*, vol. 16, no. 3, 2001.

[13] S. M. Easterbrook, J. Singer, M.-A. D. Storey, and D. E. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, Ed. Springer London, 2008.

[14] O. Hummel, "Transparente Bewertung von Softwaretechnik-Projekten in der Hochschullehre," in *Tagungsband des 13. Workshops "Software Engineering im Unterricht der Hochschulen"*, 2013, pp. 103–114.

[15] F. Fagerholm, M. Kuhrmann, and J. Münch, "Guidelines for Using Empirical Studies in Software Engineering Education," in *Software Engineering und Software Management 2018, Fachtagung des GI-Fachbereichs Softwaretechnik*, 2018, pp. 85–87.

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

[17] C. Wohlin, *Empirical Software Engineering: Teaching Methods and Conducting Studies*. Berlin, Heidelberg: Springer, 2007, pp. 135–142.

[18] M. Kuhrmann, "Teaching Empirical Software Engineering Using Expert Teams," in *Tagungsband des 15. Workshops "Software Engineering im Unterricht der Hochschulen*, 2017, pp. 20–31.

We acknowledge helpful comments from our study dean Markus Westner.