

GESELLSCHAFT
FÜR INFORMATIK



Steffen Becker, Christian Gerth (Hrsg.)

Proceedings der SEUH 2023

**23.-24. Februar 2023
Paderborn, Deutschland**

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings

Series of the Gesellschaft für Informatik (GI)

Volume P- [333](#)

ISBN 978-3-88579-727-2

ISSN 1617-5468

Volume Editors

Steffen Becker & Christian Gerth

Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany /

Hochschule Osnabrück, Albrechtstr. 30, 49076 Osnabrück, Germany

steffen.becker@iste.uni-stuttgart.de / c.gerth@hs-osnabrueck.de

Series Editorial Board

Andreas Oberweis, KIT Karlsruhe,

(Chairman, andreas.oberweis@kit.edu)

Torsten Brinda, Universität Duisburg-Essen, Germany

Dieter Fellner, Technische Universität Darmstadt, Germany

Ulrich Flegel, Infineon, Germany

Ulrich Frank, Universität Duisburg-Essen, Germany

Michael Goedicke, Universität Duisburg-Essen, Germany

Ralf Hofestädt, Universität Bielefeld, Germany

Wolfgang Karl, KIT Karlsruhe, Germany

Michael Koch, Universität der Bundeswehr München, Germany

Peter Sanders, Karlsruher Institut für Technologie (KIT), Germany

Andreas Thor, HFT Leipzig, Germany

Ingo Timm, Universität Trier, Germany

Karin Vosseberg, Hochschule Bremerhaven, Germany

Maria Wimmer, Universität Koblenz-Landau, Germany

Dissertations

Rüdiger Reischuk, Universität Lübeck, Germany

Thematics

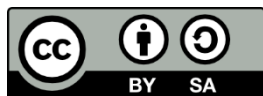
Agnes Koschmider, Universität Kiel, Germany

Seminars

Judith Michael, RWTH Aachen, Germany

© Gesellschaft für Informatik, Bonn 2023

printed by Köllen Druck+Verlag GmbH, Bonn



This book is licensed under a Creative Commons BY-SA 4.0 licence.

Vorwort

Herzlich Willkommen zum Tagungsband der Konferenz zum Software Engineering im Unterricht der Hochschulen (SEUH) 2023. Der Tagungsband bietet einen Überblick über die Einreichungen und eine Zusammenfassung der Themen.

Die SEUH ist seit vielen Jahren das Forum im deutschsprachigen Raum, auf dem Lehrende aus Universitäten, Hochschulen für angewandte Wissenschaften sowie dualen Hochschulen ihre Erfolge, Misserfolge und Erfahrungen in der Software Engineering Ausbildung vorstellen, diskutieren und gemeinsam die Qualität der Lehre verbessern. Auch die Aus- und Weiterbildung von Praktikern steht aufgrund der aktuellen Entwicklungen in der Wirtschaft im Fokus der SEUH. Neben inhaltlichen Impulsen rund um Lehren und Lernen bietet die SEUH viel Raum für Diskussion und den gegenseitigen Austausch. Viele Lehrende haben von der SEUH Impulse für ihre Arbeit erhalten.

Die SEUH 2023 wurde durchgeführt an der Uni Paderborn, zeitlich abgestimmt mit der Tagung Software Engineering (SE) der Gesellschaft für Informatik. Die SEUH lebt vom intensiven kollegialen Austausch. Um diesen zu stärken wird neben den üblichen Vortragsformaten auch ein Diskussionsforum integriert, in dessen Rahmen aktuell für die Teilnehmer:innen relevante Themen gemeinschaftlich bearbeitet werden.

Im Fokus der SEUH 2023 standen die Themenbereiche Aus- und Weiterbildung im Beruf sowie Intelligente Tutorsysteme, KIs für die Ausbildung in SE. Beispiele für mögliche Themen im Bereich Aus- und Weiterbildung im Beruf waren definiert als Wiederverwendbare Software, Komponenten, Architekturen, Moderne Softwarequalitäten (u.a., Skalierbarkeit, Resilienz), Technikfolgenabschätzung, Ethische Aspekte des Software Engineerings, Gesellschaftliche Aspekte des Software Engineerings, Neue Programmiersprachen und -paradigmen sowie unterstütztes Lebenslanges Lernen.

Im Vorfeld der Konferenz schlug passend zum Konferenzmotto das ChatGPT Sprachmodell hohe Wellen und zeigte, wie aktuell die Themengebiete der Konferenz gewählt waren. Nicht nur kann ChatGPT Prüfungsaufgaben lösen, es kann auch ganze Klausuren generieren. Hier bieten sich zukünftig vielfältige Chance und Risiken, die auf der SEUH 2023 thematisiert wurden.

Die angenommenen Papers widmen sich diesen Themen von verschiedenen Standpunkten aus. Die Einreichungen wurden dabei in die drei Sessions gegliedert, in denen sie auch auf der SEUH vorgestellt und diskutiert wurden.

Wir danken allen, die zum Gelingen der SEUH 2023 beigetragen haben. Insbesondere danken wir den Autoren, ohne deren Beiträge die Konferenz gar nicht möglich gewesen wäre, den Gutachten für die reibungslose Erstellung ihrer Reviews, den Keynote-Speakern für ihre anregenden Vorträge, den Sponsoren, der Universität Paderborn, und der GI e.V. mit dem Fachbereich Softwaretechnik.

Wir hatten eine abwechslungsreiche Konferenz mit vielen Beiträgen aus allen Dimensionen der Softwaretechnikausbildung sowie viele spannende Anregungen und neue Ideen.

Paderborn, im Februar 2023

Steffen Becker, Christian Gerth

Tagungsleitung

Steffen Becker
Christian Gerth

Universität Stuttgart
Hochschule Osnabrück

Programmkomitee

Martin Fränze
Michael Goedicke
Anne Koziol
Stephan Krusche
Karsten Morisse
Markus Müller-Olm
Rick Rabiser
Axel Schmolitzky
Juliane Siegeris
Veronika Thurner
Thomas Vogel
Karin Vosseberg

Universität Oldenburg
Universität Duisburg-Essen
Karlsruhe Institute of Technology
Technische Universität München
Hochschule Osnabrück
Universität Münster
Johannes Kepler Universität Linz
Hochschule für Angewandte Wissenschaften Hamburg
Hochschule für Technik und Wirtschaft Berlin
Hochschule München
Universität Paderborn
Hochschule Bremerhaven

Ständiges Organisationskomitee

Steffen Becker
Axel Schmolitzky
Veronika Thurner

Universität Stuttgart
HAW Hamburg
Hochschule München

Inhaltsverzeichnis

SEUH 2023

Session 1

Kerstin Jacob, Daniel Hallmann, Gerald Lüttgen, Vanessa Völpe <i>On Developing an E-Assessment Tool for Agile Practices</i>	13
Julia Krumme, László Kovács, Alexandra Teynor <i>Ethik in der SWE-Ausbildung</i>	19
Anna Sabine Hauptmann <i>Soziale Kompetenz in Projekten</i>	25

Session 2

Sandro Speth <i>Teaching Object-Oriented Programming with the Objects-first Approach</i> .	35
Henning Schlender, Ralf Stemmer, Kim Grüttner, Günter Ehmen, Bernd Westphal, Friederike Bruns <i>Teaching Cyber-Physical Systems in Student Project Groups: Alumni Assessment</i>	43
Axel Böttcher, Veronika Thurner <i>Digitale Prüfungen für Softwareentwicklung</i>	55
Oliver Radfelder, Karin Vosseberg <i>Nachhaltigkeit als Qualitätskriterium von Software</i>	67

Session 3

Philipp Felix Schmeier, Stefan Bente
The Microservice Dungeon: Realitätsnahe Lehre komplexer Softwarearchitekturen 79

Axel Schmolitzky, Henri Bureau
OPPSEE – eine Online-Plattform zum Programmieren Üben 93

Niklas Meißner, Sandro Speth, Uwe Breitenbücher
An ITS Concept for a Gamified e-Learning Platform for Higher Computer Science Education 105

Autorenverzeichnis

SEUH 2023

Session 1

On Developing an E-Assessment Tool for Agile Practices

Kerstin Jacob,¹ Daniel Hallmann,¹ Gerald Lüttgen,¹ Vanessa Völpe²

Abstract: As agile software development processes are widely applied in industry, students need to develop a good understanding of agile principles and practices as part of their education. During our ten years of experience with teaching the Scrum framework in team project modules, we noticed that students frequently struggle with writing good user stories and concise sprint goals. To support students in mastering basic practices and to free limited teaching resources for more in-depth discussions on advanced topics, e-assessment tools should be developed to provide timely feedback on at least common mistakes. This paper argues that, and sketches how, research on quality criteria for artifacts of agile development can serve as the basis for such a tool. However, multiple challenges remain that we invite lecturers and researchers to discuss.

Keywords: agile software development; e-assessment tool; Scrum

1 Introduction

Agile development processes are ubiquitous in the software industry today. Therefore, it is highly relevant for computer science students to develop a thorough understanding of agile principles and practices during their software engineering education. To meet this need, our Chair replaced the waterfall model with a Scrum-based model [SS20] in student team projects about ten years ago. In particular, the Bachelor project modules cover multiple skills beyond programming practices, such as discussing requirements with stakeholders, casting requirements into user stories, and estimating user story sizes and priorities. Giving students the practice they need in order to learn these skills and to build a habit of agile thinking requires time and intensive mentoring by lecturers.

We have observed that a vast amount of the time spent in supervision meetings with students is devoted to the same basic practices: phrasing of user stories and sprint goals, and splitting software features sensibly into user stories. Not only do students struggle with these topics, but professionals do, too [Co04]. Thus, it is important to find ways to support students with the writing of these artifacts, and provide formative feedback [Sh08]. Because contact hours between lecturers and students are limited due to the difficulty of finding sufficiently many tutors, we believe that e-assessment tools are needed that identify at least the more shallow mistakes and highlight them to students.

However, generating feedback on sprint goal phrasing and user story writing in an automated manner is challenging due to the diverse and often unstructured textual nature of these

¹ University of Bamberg, Software Technologies Research Group, Germany, firstname.lastname@uni-bamberg.de

² University of Bamberg, Germany, vanessa@voelpe.de

artifacts of interest. While it is difficult to define clear and measurable quality criteria, there are multiple guidelines (see, e.g., [Co04; Je01; Wa03]) derived from practical experience that have been taken up and evaluated by the scientific community (see, e.g., [Ha20; Lu16]) and that may provide a basis for the desired e-assessment tool.

This paper states our vision for the e-assessment tool on agile practices (Sect. 2) and reports on our current prototyping efforts (Sect. 3). Our approach is built upon natural language processing (NLP) techniques [JM09] and informed by recent research on measuring user story quality [Ha20]. While our tool is not yet complete and has not been deployed in our teaching, we encountered multiple challenges and open research questions during prototyping (Sect. 4) to which we believe the SEUH community can contribute.

2 Vision

We introduce the concepts of agile practices in an introductory lecture module to software engineering and practice the writing of user stories and sprint goals with students in accompanying practicals. Additionally, we start off our project modules with tutorials on these practices. Nevertheless, students in our team projects still struggle with the writing of user stories and sprint goals throughout the semester.

<p>User Story 1: Database Implementation</p> <p>As a user, I want to save and load things to and from a database.</p> <p>The story is done, when</p> <ul style="list-style-type: none"> - a current plan can be saved on a MariaDB - all entities are correctly related to each other and the relations in the database reflect the planned design of the developers 	<p>Sprint goal:</p> <p>Implement the basic functionalities, such as database connection, JSON imports and the basic view so that we have a reliable, stable foundation on which we can build and so that we will be able to deliver a high quality product.</p>
---	--

Fig. 1: Exemplary user story and sprint goal suggested by a student team.

Fig. 1 (left) shows an exemplary user story from one of our student teams, which includes multiple common mistakes: the user story misses the rationale ([why]), and thus does not fully follow the pattern “[title]–As [persona], I will [what] so that [why]–[acceptance criteria]–[attachments]”¹. The story does not describe a vertical application slice, but concerns only the database connection and uses technical terms such as “MariaDB” instead of the language of the customer. Additionally, the story includes unspecific terms such as “things” and is partly phrased in an unreadable manner (e.g., the long second acceptance criterion).

¹ <https://www.agilealliance.org/glossary/user-story-template> (last accessed: 28 Oct 2022)

User stories are a central artifact based on which a team estimates effort, creates an architecture, builds the system, and sets up the test strategy. Thus, a poorly phrased user story can lead to lengthy discussions and divergent views between team members, which increases the risk of incorrect feature implementation and expensive rework [Co04].

Similar to user stories, a well-phrased sprint goal is essential to communicate to stakeholders why the sprint is valuable, and to provide focus in the sprint meetings. However, we observe that students stumble over formulating sprint goals. Fig. 1 (right) shows a sprint goal as suggested by one of our student teams. It contains mistakes similar to the aforementioned user story: unspecific phrasing (e.g., “Implement the basic functionalities”), low readability (e.g., due to the long nested sentence), and usage of technical keywords (e.g., “JSON imports”).

We envision an e-assessment tool that supports the students by detecting such common mistakes and giving formative feedback with actionable suggestions on how to improve the artifacts. While summative feedback is given at the end of a module to assess the students’ learning outcomes, formative feedback is provided to the students throughout the module to improve their learning process. Because a sprint in our project module is only two weeks long and included in the normal semester schedule with limited hours each week, it is very important that students receive feedback on artifacts in a timely manner so that they can improve them while the sprint is still in progress. This is difficult with supervision sessions that are only conducted weekly, but can likely be accomplished by an e-assessment tool. The application of the e-assessment tool shall leave more time in the supervision meetings for discussing deeper aspects of agile practices, such as user story splitting and size estimation, and for reflecting on the underlying agile principles.

3 Approach

Our current e-assessment prototype receives user stories and sprint goals as well as meta-data such as authors and creation date from our process management tool (GitLab²), and outputs an email with textual feedback to the artifacts’ authors. The tool’s quality measurements are based on recent research [Ha20] on the quality of artifacts of agile development. In particular, the measurements use NLP techniques [JM09] to analyze textual information, and GitLab meta-data such as revision counts. This section details the employed quality criteria, their measurement, and the feedback generation.

3.1 Quality Criteria and their measurement

There exists a variety of guidelines that can aid authors in writing high-quality user stories, e.g., Cohn’s guidelines [Co04], the INVEST criteria by Wake [Wa03], or CCC (Card,

² <https://about.gitlab.com/solutions/agile-delivery/> (last accessed 28 Oct 2022)

Conversation, Confirmation) [Je01], which have predominantly been derived by consultants from their practical experience. An empirical study [Lu16] found that practitioners consider the principles helpful for creating a shared understanding and increasing productivity within the team, although the principles are highly qualitative and thus difficult to assess automatically.

Recently, Hallmann [Ha20] defined quality criteria for user stories, which are unique in that their measurements can be automated. However, the practical utility and validity of this work has only been assessed preliminarily. To address this shortcoming, and in parallel to our e-assessment prototype, we have initiated an expert study with practitioners from industrial development projects to evaluate Hallmann's criteria and their measurements. Hallmann has selected the following four quality criteria based on a literature review and brainstorming sessions with experts:

Complete: This quality criterion captures whether all form fields of the user story template by Connextra³ have been filled.

Readable: This quality criterion measures the readability of a given user story using the "Flesch reading ease" [Fl48], which estimates the readability of a document based on the average sentence length and word length.

Valuable: This quality criterion describes the business value that a user story represents. The measurement is focused on the usage of domain language and calculates the percentage of domain words relative to the story's total number of words.

Saturated: A user story should be the basis of discussion between the development team and the stakeholders, and also within the team, and be continuously refined. This quality criterion depicts how often such refinements have taken place, based on data retrieved from the employed process management tool.

Measuring the stated criteria is non-trivial due to the semi-structured (in the case of user stories) and unstructured (sprint goals) text formats. To measure the quality criterion *complete*, we have implemented a pattern matcher using the NLP libraries *spaCy*⁴ and *sklearn*⁵ to retrieve the individual parts such as the persona or the rationale from a user story written in German or English. Pattern matching typically succeeds as long as the text rather strictly follows the template. However, it quickly fails if the structure of the inputs differs from the expected pattern, e.g., given sentence constructs such as "I, as an experienced user" or "As a doctor or pharmacist". Thus, we have additionally experimented with a supervised learning approach using a tokenizer, a vectorizer, and a support vector machine as a classifier, with promising first results.

Similarly, we employ a pattern matcher to measure the quality criterion *valuable*, which

³ <https://www.agilealliance.org/glossary/user-story-template> (last accessed: 28 Oct 2022)

⁴ <https://spacy.io/usage/spacy-101> (last accessed: 28 Oct 2022)

⁵ <https://scikit-learn.org/stable/> (last accessed: 28 Oct 2022)

relies on finding domain-specific terms in the story text. To identify such terms, we use a vetted glossary generated during the student project blastoff. In addition to measuring the degree of domain words as is done in [Ha20], we also identify technical keywords using a glossary based on the technology stack of our project modules. To calculate the sentence and word length for the quality criterion *readable*, we employ the sentencizer, tokenizer and syllables annotator of the *spaCy* library. Criterion *saturated* is calculated from retrieved GitLab meta-data, and thus does not require NLP techniques.

While Hallmann [Ha20] considers each user story in isolation, we have added a quality criterion *independent*, which calculates the semantic similarity for each pair of user stories in a given set; user stories with high similarity might be duplicates or not functionally independent. We have done so because student teams do, at least at the start of the semester, seldom act as a team and several students come up independently with user stories, which unsurprisingly, are often neither orthogonal nor complementary.

As suggested by the example sprint goal in Fig. 1 (right), the quality criteria *readable* and *valuable* can also be applied to sprint goals. Nevertheless, it is an open research problem how to measure whether sprint goals are phrased in a way that motivates development teams and fosters focus during Scrum meetings.

3.2 Feedback Generation

Interpreting and presenting the results of our measurements for a particular user story or sprint goal, and thus providing meaningful formative feedback, has proven to be particularly challenging. There is a large body of research surveyed in [Sh08] which is concerned with how to provide feedback to students wrt. content, phrasing, timing, and frequency. Our e-assessment prototype currently reports compliance or deviation from the above criteria in an email sent to the artifacts' authors. Evaluating our prototypical feedback generator in a user study with students, and enhancing its feedback capability, is ongoing work.

Unclear value to the user

Your user story has been checked for the quality criterion *valuable*. We detected the use of the technical terms *database* and *MariaDB*, and calculated a domain terminology degree of 0%.

Recall that the aim of a story is to convey value to the user. Thus, the story should be written in the language of the user and avoid technical terms.

Fig. 2: Exemplary feedback for the quality criterion *valuable*.

Fig. 2 shows an exemplary feedback for the quality criterion *valuable*. It consists of a short description of the identified problem, including the concrete markers in the text on which the feedback is based. Additionally, the report explains why following the quality criterion is relevant.

4 Conclusions

Teaching an agile framework like Scrum in a team project module requires timely and constant feedback to students. Due to the scarcity of teaching resources, we developed a prototypical e-assessment tool for automating this feedback process for common, but not semantically deep mistakes made by students when writing user stories and sprint goals. The tool is based on NLP techniques [JM09] and recent research on quality indicators for user stories [Ha20]. Before it can be deployed in teaching, its feedback generator needs to be optimized. It should also be evaluated user stories and sprint goals written in past student team projects.

Multiple questions and challenges remain, which we invite lecturers, researchers, and practitioners to discuss:

- (1) Which agile practices are most difficult for students (and practitioners) to master? Which of the practices can be evaluated by an e-assessment tool?
- (2) What is a sound methodology for deriving quality criteria from agile artifacts? How can such criteria be measured, and how should measurements be interpreted?
- (3) How can e-assessment tools be integrated best in a module's teaching concept? When should feedback be distributed?

References

- [Co04] Cohn, M.: User Stories Applied: For Agile Software Development. Addison-Wesley Professional, 2004.
- [Fl48] Flesch, R.: A New Readability Yardstick. *J. Appl. Psychol.* 32/3, p. 221, 1948.
- [Ha20] Hallmann, D.: "I Don't Understand!": Toward a Model to Evaluate the Role of User Story Quality. In: *XP*. Pp. 103–112, 2020.
- [Je01] Jeffries, R.: *Essential XP: Card, Conversation, Confirmation*, 2001, URL: <https://www.ronjeffries.com/xprog/articles/expcardconversationconfirmation>.
- [JM09] Jurafsky, D.; Martin, J. H.: *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Pearson Prentice Hall, 2009.
- [Lu16] Lucassen, G.; Dalpiaz, F.; v. d. Werf, J.; Brinkkemper, S.: The Use and Effectiveness of User Stories in Practice. In: *REFSQ*. Springer, pp. 205–222, 2016.
- [Sh08] Shute, V. J.: Focus on Formative Feedback. *Review of Educational Research* 78/1, pp. 153–189, 2008.
- [SS20] Schwaber, K.; Sutherland, J.: *The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game*, 2020, URL: <https://scrumguides.org>.
- [Wa03] Wake, B.: *INVEST in Good Stories, and SMART Tasks*, 2003, URL: <https://www.xp123.com/articles/invest-in-good-stories-and-smart-tasks>.

Positionspapier: Ethik in der Ausbildung für Software-Entwickler:innen

Ethische und soziale Implikationen der Digitalisierung

Julia Krumme¹, László Kovács², Alexandra Teynor³

Abstract: Ethische und soziale Aspekte von Softwareprojekten werden in der aktuellen Informatikausbildung kaum beachtet. Während es mittlerweile hinreichend bekannt ist, dass erfolgreiche Projekte interdisziplinäre Teams aus Softwareentwickler:innen, Gestalter:innen und Fachexpert:innen aus der Anwendungsdomäne benötigen, werden die Teams selten um Fachrichtungen aus dem geisteswissenschaftlichen Spektrum ergänzt. In vorliegendem Beitrag werden konkrete Vorschläge zur Ergänzung der Informatikausbildung gemacht, damit die künftigen Entwickler:innen befähigt werden, den ethischen und sozialen Herausforderungen der Digitalisierung besser zu begegnen.

Keywords: Ethik; Digitalisierung; Software Engineering Ausbildung

1 Problemstellung

Die »digitale Revolution« [SvW17] sorgt innerhalb der gesellschaftlichen Strukturen für Umbrüche, die mitunter mit denjenigen vergleichbar sind, die durch die Industrialisierung im 19. Jahrhundert hervorgerufen wurden. Mit dem Einzug digitaler Techniken in weite Bereiche des Lebens (sei es in der medizinischen Versorgung, bei der verbesserten Sicherheit im Verkehr oder zur Ermöglichung von mehr Teilhabe für Menschen mit Einschränkungen), findet ein tiefgreifender Wandel für Mensch und Gesellschaft statt. Der Fortschritt sorgt allerdings nicht nur dafür, dass unser Leben schneller, effizienter und bequemer wird; er verändert es gleichzeitig grundlegend. Die Art und das Ausmaß dieser Veränderung wird allerdings bislang nur unzureichend erfasst und reflektiert, insbesondere innerhalb derjenigen technischen Disziplinen, die diesen technologischen Fortschritt hervorbringen. Eine stichprobenartige Literatur-Recherche in den Datenbanken Scopus, PhilPapers, ACM und IEEE zeigte, dass in den letzten 23 Jahren von knapp 1.000.000 Papers zum Thema »software development« insgesamt nur 173 Publikationen verzeichnet sind, die in Titel, Abstract oder Keywords die Suchbegriffe »ethics« und »software development« aufweisen.

¹ Hochschule Augsburg, Fakultät für angewandte Geistes- und Naturwissenschaften, An der Hochschule 1, 86161 Augsburg, Deutschland julia.krumme@hs-augsburg.de

² Hochschule Augsburg, Fakultät für angewandte Geistes- und Naturwissenschaften, An der Hochschule 1, 86161 Augsburg, Deutschland laszlo.kovacs@hs-augsburg.de

³ Hochschule Augsburg, Institut für agile Softwareentwicklung, Fakultät für Informatik, An der Hochschule 1, 86161 Augsburg, Deutschland alexandra.teynor@hs-augsburg.de

Insbesondere, so zeigt die Erfahrung der Lehrpersonen an der Hochschule Augsburg, legen zukünftige Entwickler:innen oft den Fokus darauf, was (technisch) möglich ist, und weniger darauf, welche Konsequenzen aus einer Entwicklung folgen oder wie diese die Bedingungen des Sozialen und des individuellen Lebens verändern. Das Zur-Verfügung-haben und mögliche Vernetzen von großen Datenmengen (*Big Data*) wie auch die Entwicklungen im Bereich der künstlichen Intelligenz (KI), berühren Themenfelder, die in den technischen Fachdisziplinen weder isoliert bearbeitet noch hinreichend reflektiert werden können, einfach weil das nötige Werkzeug dafür (noch) nicht zur Verfügung steht.

Aus dieser Perspektive erscheint es geradezu zwingend, die Ausbildung – insbesondere von Softwareentwickler:innen – zukünftig dahingehend zu erweitern, dass den ethischen und sozialen Herausforderungen der Digitalisierung angemessen begegnet werden kann.

2 Multidisziplinäre Ausbildung aktuell unvollständig

In der Softwareentwicklung ist bereits seit geraumer Zeit bekannt, dass verschiedene Expert:innen gemeinsam an einen Tisch gebracht werden müssen, um Projekte erfolgreich entwickeln zu können [LL10]. Dazu zählen neben den Techniker:innen und Personen aus gestalterischen Disziplinen, z.B. UX- oder UI-Design, auch die Fachexpert:innen aus der Anwendungsdomäne. Was allerdings bisher kaum in den Fokus gerückt wurde, ist eine »Erweiterung« der Multidisziplinarität um Fachwissen aus dem geistes- und sozialwissenschaftlichen Spektrum.

Dabei ließen sich insbesondere durch eine interdisziplinäre Verzahnung von Technik, Gestaltung und Ethik (sowohl Individualethik als auch Sozialethik) neue Denk-Räume eröffnen, die eine praktische Auseinandersetzung mit zentralen Fragen der Digitalisierung und den sozio-kulturellen Wechselwirkungen in die Gesellschaft hinein ermöglichen. Auf diese Weise ließe sich den Herausforderungen der Digitalisierung bereits dort begegnen, wo sie entstehen: In der Entwicklung von Technik.

Durch eine multidisziplinäre Zusammenarbeit bereits in der Ausbildung können alle beteiligten Disziplinen profitieren: Während z.B. Entwickler:innen lernen, ihren Blick von der reinen Machbarkeit stärker auf normative Aspekte lenken, profitieren geisteswissenschaftlich ausgerichtete Studiengänge davon, wenn sie die technische Grundlagen ihrer theoretischen Überlegungen kennen- und verstehen lernen. So kann es sehr hilfreich sein, wenn z.B. Ethiker:innen auch ein *technisches* Verständnis davon entwickeln, was gemeint ist, wenn von KI (künstlicher Intelligenz) die Rede ist, ohne sich vorschnell auf abstrakte ontologische Positionen aus der theoretischen Philosophie zurückzuziehen, die mit dem Alltag der technischen Entwicklung nicht immer Hand in Hand gehen.

Natürlich existieren bereits zahlreiche *Guidelines* und *Codes of Conduct* die eine ethisch informierte Entwicklung und Funktion von Technik sicherstellen sollen. So werden allein in der Übersichtsarbeit von Jobin [JIV19] – dort eingegrenzt auf das thematische Feld der KI –

84 *Guidelines* bzw. *Codes* erwähnt, die ethische Prinzipien beschreiben. Allerdings sind diese mit einer Schwierigkeit behaftet, die sich im übrigen für alle »mittleren Prinzipien« angewandter Ethiken ergibt: Sie sind hinreichend abstrakt. Das ist einerseits notwendig, um überhaupt eine übergreifende Beschreibung ähnlicher (aber eben nicht identischer) Einzelfälle möglich zu machen. Andererseits ist es genau dieses Abstraktions-Level, das einen Transfer des ethischen Prinzips auf den konkreten Einzelfall in der Realität schwierig gestalten kann. In der Folge sind die relevanten *Codes of Conduct* zwar bekannt, finden aber oftmals in der Praxis nicht oder nur unzureichend Anwendung, einfach, weil sie sich nicht automatisch in reale Entwicklungskontexte übersetzen bzw. übertragen lassen. (vgl. [Mi19] [Pr21])

Um diesem *Theorie-Praxis-Gap* zu begegnen und ethisch informierte, gut begründete Lösungen finden zu können, sind praktische, deliberative Fertigkeiten notwendig, die über die bisher in der Ausbildung von technischen Berufen vermittelten Inhalte hinausgehen.

3 Die Rolle der angewandten Ethik

Angewandte Ethik als integrierter Teil einer multidisziplinären Praxis kann dabei helfen, die notwendigen Werkzeuge für den Transfer von Prinzipien in den konkreten Anwendungsfall bereitzustellen. Nimmt man exemplarisch den Bereich der Software-Entwicklung in den Blick, so ließe sich ein entsprechendes Verfahren gut in agile Entwicklungsmethoden (zum Beispiel Scrum) einbinden. (vgl. [Zu22]). Ein Aufdecken potentieller (ethischer) Probleme und Konflikte könnte so strukturell in den Entwicklungsprozess eingebunden werden, dass eine kritische Evaluation von Software und ihrer Anwendungskontexte nicht erst nach der Fertigstellung erfolgt. Deliberative Prozesse und kritische Reflektion bedürfen dazu einer gewissen Übung und erfordern ergänzende Kenntnis z.B. über verschiedene Arten von moralischen Problemen (seien es nun Konflikte zwischen moralisch Gebotenem und außermoralischen Handlungsanreizen, zwischen verschiedenen Werten, oder aber Konflikten, die sich daraus ergeben, dass eine gewisse Unsicherheit darüber besteht, welche empirischen Fakten des konkreten Anwendungsfalls überhaupt für eine normative Bewertung relevant sind). Dies deutet darauf hin, dass für eine wirkungsvolle Einbettung zumindest dreierlei vonnöten ist: Zum einen die Fähigkeit zur möglichst umfassenden Analyse des jeweiligen Handlungs- und Entwicklungskontextes, zum anderen praktische Erfahrung in deliberativen Verfahren, die alle Beteiligten in die Lage versetzt, sich mit zwei – zusammenhängenden – Problemen auseinandersetzen zu können, nämlich der Identifikation möglicher moralischer Problemstellen und, drittens, die argumentative Bearbeitung derselben, die dann schließlich in begründeten Maßnahmen für die Praxis mündet.

4 Digitalisierungskollegs als gangbare Antwort

Um auf die komplexen Fragen, die die Digitalisierung an die Gesellschaft stellt, in Zukunft angemessen antworten zu können, ist es notwendig, dass die bisher schon existierenden *Codes*

of Coduct und *Guidelines* für technische Berufe eine praxisgebundene Ergänzung erfahren. Ein gangbarer Weg scheint hier die Verknüpfung von technischer Ausbildung mit praktischen Verfahren aus den Sozial- und Geisteswissenschaften, wie sie zum Beispiel an Hochschulen für Angewandte Wissenschaften und Universitäten durch eine engere interdisziplinäre Verzahnung von Studiengängen geleistet werden könnte. So ließen sich entsprechende ethische Verfahren in Entwicklungsprozesse einbinden, damit Studierende technischer Fachrichtungen sich das notwendige Know-how aneignen können, um moralische Probleme, die im Zusammenhang mit ihrer Arbeit stehen, zu erkennen und um zu begründeten und damit nachhaltigen Lösungen zu gelangen.

Erste Ansätze, eine solche Verzahnung der Fachbereiche vorzunehmen, zeigen sich heute u.a. in der Einrichtung verschiedener »Digitalisierungskollegs« an Hochschulen in Bayern, die als prototypische Ansätze gesehen werden können, dem durch die Digitalisierung angestoßenen Wandel zukünftig zu begegnen. Gleichzeitig besteht in diesem Bereich weiterhin großer Forschungsbedarf: Dabei sollte es nicht nur um inhaltliche Fragen gehen, die den Austausch und die Vernetzung von mehreren, doch sehr unterschiedlichen Disziplinen betrifft, sondern auch um didaktische Herausforderungen, die sich durch eine derartige Verbindung ergeben.

5 Mögliche Ausgestaltung eines Digitalisierungskollegs

Exemplarisch lässt sich ein solches Digitalisierungskolleg anhand des Konzepts der Hochschule für Angewandte Wissenschaften Augsburg (HS Augsburg) veranschaulichen (siehe Abbildung 1). Dort soll durch 3 Ausbildungsschwerpunkte den verschiedenen Abschnitten im Studienverlauf sowie der Disparität der einzelnen Studiengänge Rechnung getragen werden.

Den Rahmen bilden in allen Fällen innovative IT-Anwendungen, die bereits im Entwicklungsprozess reflexiv begleitet und auf ihre gesellschaftlichen Auswirkungen und normativen Implikationen hin untersucht werden sollen. Dafür eignen sich in besonderer Weise diejenigen Projekte, die im Rahmen von Projektarbeiten durch Studierende in Kooperation mit externen Projektpartnern (meist KMUs aus der Region) entwickelt werden ebenso wie Abschlussprojekte, vor allem in Master-Studiengängen aus den Fachbereichen der Informatik und der Gestaltung, sofern sie einen Bezug zur Digitalität aufweisen.

Diese Projekte können dann einerseits

1. *ex ante* d.h. bereits *während* der Entwicklung, reflexiv begleitet werden, was für die Projekte wie auch für Abschlussarbeiten gleichermaßen zutreffen kann, wie auch andererseits
2. *ex post*, d.h. *nach* Fertigstellung der ersten Prototypen, hinsichtlich ihres Einflusses auf Gerechtigkeit und gutes Leben in der Gesellschaft befragt werden. Dieses *ex-post*-Vorgehen wird vor allem für Abschlussarbeiten relevant sein, bei denen die

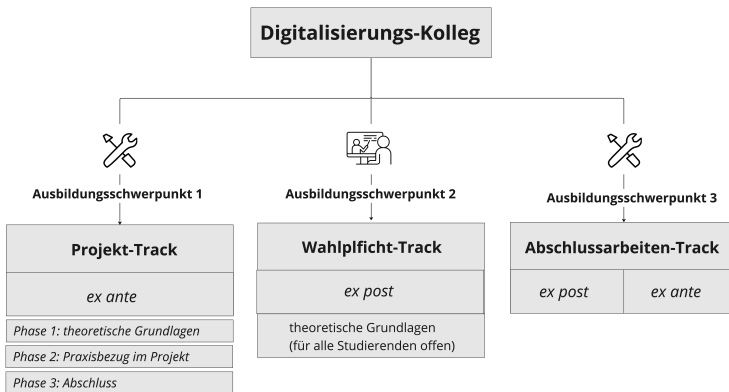


Abb. 1: Vorgeschlagene Ausbildungsschwerpunkte

Entwicklung der technischen Grundlage bereits abgeschlossen ist, welche dann nachgängig durch eine reflexive Stufe ergänzt wird.

Der **erste Ausbildungsschwerpunkt** ist an ein Praxisprojekt geknüpft, dessen Laufzeit ein oder zwei Semester beträgt. Dabei werden drei Phasen durchlaufen (vgl. Abbildung 1): Theoretische Grundlegung, praktische Umsetzung und Abschluss-Präsentation. Über den gesamten Zeitraum arbeiten die Studierenden in multidisziplinären Teams zusammen und werden dabei von Personen betreut, die die entsprechende Expertise der angewandten Ethik und den Geisteswissenschaften mitbringen.

In einem **zweiten Ausbildungsschwerpunkt** können Studierende *aller* Fakultäten im Rahmen von Vorlesungen, Seminaren und Kompaktkursen ECTS-Punkte erwerben. Hierbei liegt der Schwerpunkt auf den theoretischen Grundlagen, die unabhängig von der Durchführung eines Praxisprojekts sind.

Der **dritte Ausbildungsschwerpunkt** kann so ausgestaltet werden, dass Studierende ihre Abschlussarbeiten im Rahmen von Projekten vorbereiten und/oder durchführen können. Sie werden dabei in der Erarbeitung der notwendigen theoretischen Grundlagen individuell betreut und erhalten die Möglichkeit, die sozio-kulturellen Wechselwirkungen und ethischen Aspekte ihrer spezifischen Fragestellung gemeinsam mit den interdisziplinären Teams aus dem ersten Ausbildungsschwerpunkt zu diskutieren und kritisch zu reflektieren.

6 Erwartungen und Ausblick

Ziel des Digitalisierungskollegs ist es, in der theoretischen Fundierung, insbesondere aber in der praktischen Anwendung die Studierenden dazu zu befähigen, sich den ethischen

Fragestellungen und gesellschaftlichen Auswirkungen kritisch zu stellen, die sich durch die Digitalisierung im Allgemeinen und durch die jeweiligen konkreten Projekte im Besonderen ergeben.

Über Vorlesungen, Seminare und Workshops mit ethischem Schwerpunkt, die sowohl im Wahlpflichtprogramm wie auch in den Pflichtfächern der Studiengänge verankert werden, können Studierende technischer Fachrichtungen ihren Wissenshorizont über die eigene Disziplin hinaus erweitern. Sie lernen neue Grenzflächen der Technik kennen und eignen sich Methoden an, um sich kritisch mit den ethischen Aspekten, die mit der Digitalisierung zusammenhängen, auseinandersetzen zu können.

Gleichzeitig wird die dort gelernte Theorie direkt in konkreten Praxisprojekten erprobt. Multidisziplinäre Teams arbeiten so in der direkten diskursiven Auseinandersetzung daran, ein Bewusstsein für die ethischen Dimensionen zu entwickeln und treten in einen Diskurs über mögliche Auswirkungen ein. Wenn dabei sicher auch nicht *alle* Herausforderungen der Digitalisierung erkannt oder gar abschließend behandelt werden können, so erachten wir es dennoch als wertvoll, ethische Aspekte in den Entwicklungsprozess zu integrieren, anstatt nur nachträglich feststellen zu können, was *nicht* gemacht hätte werden sollen.

Literaturverzeichnis

- [JIV19] Jobin, Anna; Ienca, Marcello; Vayena, Effy: The global landscape of AI ethics guidelines. *Nature Machine Intelligence*, 1(9):389–399, September 2019.
- [LL10] Ludewig, Jochen; Lichter, Horst: *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag, 2010.
- [Mi19] Mittelstadt, Brent: Principles alone cannot guarantee ethical AI. *Nature Machine Intelligence*, 1(11):501–507, November 2019.
- [Pr21] Pretschner, Alexander; Zuber, Niina; Gogoll, Jan; Kacianka, Severin; Nida-Rümelin, Julian: Ethik in der agilen Software-Entwicklung. *Informatik Spektrum*, 44(5):348–354, Oktober 2021.
- [SvW17] Stengel, Oliver; van Looy, Alexander; Wallaschkowski, Stephan, Hrsg. *Das Ende des Industriezeitalters und der Beginn einer neuen Epoche*. Springer VS, 2017.
- [Zu22] Zuber, Niina; Gogoll, Jan; Kacianka, Severin; Pretschner, Alexander; Nida-Rümelin, Julian: Empowered and embedded: ethics and agile processes. *Humanities and Social Sciences Communications*, 9(1):191, Juni 2022.

Woher kommt die (softwaretechnologische) Zukunft?

Anna Sabine Hauptmann¹

Abstract: Da Softwaresysteme in nahezu allen Lebensbereichen eine Rolle spielen, beteiligen sich die Softwareentwickler maßgeblich an unserer Lebensgestaltung. Demzufolge sollten wir als Lehrende auf diesem Gebiet auch ethische und soziale Aspekte in den Unterricht einbeziehen, allem voran die Frage nach dem Sinn des Tuns. Während der gemeinsamen Projektarbeit auch im Studium treten durchaus Konflikte auf, deren Ursachen meist im Verborgenen liegen. Diese Konflikte lassen sich auch zurückführen auf den Umgang mit und die Ausübung von Macht bzw. Autorität. Brücken erlauben es, Erkenntnisse aus der Softwareentwicklung mit Erkenntnissen aus anderen Fachgebieten, insbesondere aus der Hirnforschung, zu verbinden und auch in den Alltag zu übertragen. Wollen wir den disruptiven Veränderungen in unserer Gesellschaft nicht ohnmächtig gegenüberstehen, dann werden wir Proaktive.

Keywords: Soziale Kompetenz; Ethik; Macht in Projekten; Haltung; Autorität; Reflexion; Annahmen

1 Einleitung

Technischer Fortschritt und verbunden damit wirtschaftliches Wachstum sind die angestrebten Ziele in unsere Gesellschaft. Demgegenüber machen wir als Menschen gerade die Erfahrung, dass sich eine sogenannte Krise nach der anderen zeigt. Das sollte uns zum Innehalten bewegen und zum Reflektieren. Vielleicht stecken wir mit unserer hohen technischen Kompetenz bei der gesellschaftlichen Entwicklung noch in den Kinderschuhen. Bau und Einsatz von Softwaresystemen berühren, wie technische Systeme überhaupt, gleichermaßen auch ethische und soziale Aspekte. Beziehungen zwischen allen Beteiligten beeinflussen den Projekterfolg auch innerhalb der studentischen Projekte. Die Projektarbeit offenbart die Haltungen, mit der die Teilnehmer agieren, Macht und Autorität sind spürbar, nicht kommunizierte Annahmen wirken und zeigen sich oft erst am Ende des Projektes. Die Lehrveranstaltungen zum Software Engineering bieten Raum, auch diese Perspektiven zu beleuchten. Im Grunde führt dieser Perspektivwechsel auch zu Synergien. Der Beitrag gibt einigen dieser Perspektiven Raum und lädt ein, darüber zu diskutieren.

2 Macht in Projekten

An jeder Universität und Hochschule gehört Projektarbeit zum Software Engineering. Studierende arbeiten in einer Gruppe und nicht selten kommt es zu Konflikten. Welche

¹ HTW Dresden, Fakultät Informatik/Mathematik, Friedrich-List-Straße 1, 01067 Dresden, Deutschland
hauptman@informatik.htw-dresden.de

Rolle spielt Macht dabei? Was eigentlich ist Macht? Jegliche Macht ist zunächst völlig neutral; sie ist weder gut noch böse. Studierende erwarben zum Beispiel im bisherigen Studium die Kompetenz, ein Datenmodell zu entwickeln. Im Rahmen des Softwareprojektes erhält nun eine Person die Aufgabe, die erforderliche Datenbank zu modellieren und zu implementieren. Zur Kompetenz, etwas tun zu können, kommt also die Befugnis, dies auch zu dürfen. Aber welche Intention, welche Absicht steht hinter dem Tun? Diese Intention weist unseren Handlungen die Richtung. Und mit diesem dritten Aspekt verliert die Macht ihre Neutralität. Je nach Intention kann aus Macht dabei auch Machtmissbrauch werden. Die Frage ist also, mit welcher Absicht baut das beauftragte Gruppenmitglied die Datenbank. In dem betrachteten Beispiel gibt es mehrere Intentionen. Abb. 1 zeigt dies beispielhaft. Macht in Projekten hat also nicht nur etwas mit der Projektleitung zu tun.

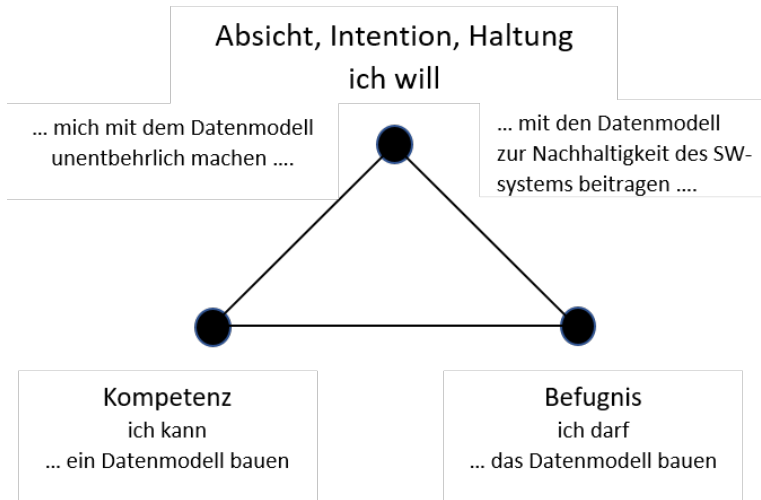


Abb. 1: Die Absicht (Haltung) gibt der Handlung ein Richtung.

Macht betrifft jede Person in der Gruppe. Und dabei ist es gleichgültig, ob diese Tatsache den Mitgliedern der Projektgruppe bewusst ist oder nicht. Das jeweilige Verhalten wirkt immer mit Konsequenzen auf die weitere Arbeit. Nimmt ein Projektmitglied seine Macht nicht wahr, ist es ohne Macht, quasi ohnmächtig. So kann Ohnmacht von außen verursacht aber auch selbst angenommen sein. Alles, was wir tun oder auch nicht tun, wird bestimmt von unserer Intention oder anders formuliert von unserer Haltung. Wir können also sagen, Softwareentwicklung ist eine Frage der Haltung wie die Lebensgestaltung überhaupt.

3 Haltung als Thema in Lehrveranstaltungen

Wenn Softwareentwicklung eine Frage der Haltung ist, dann stellt sich sofort die Frage: Wie lässt sich Haltung als wichtiger Faktor bei der Softwareentwicklung und auch beim Thema Digitalisierung vermitteln? Neurobiologisch gesehen tragen wir unsere Haltung in uns als

strukturelle Verankerung von Erfahrungen in neuronalen Netzwerken des präfrontalen Cortex im Gehirn. Die Haltung entspricht einer festen inneren Überzeugung, einer Einstellung. [TG20]

Diese quasi gespeicherte Haltung entsteht demnach aus unseren eigenen Erfahrungen; sie offenbart sich in unseren Worten, Aktionen und Reaktionen, ob es uns nun bewusst ist oder nicht. Haltung lässt sich also nicht wirklich vermitteln wie zum Beispiel die Syntax des Klassendiagrammes. Nach meiner Erfahrung ist es wichtig, zunächst auf diese Zusammenhänge hinzuweisen. Es geht darum anzuregen, die eigene Haltung zu ergründen und dabei auch zu prüfen, ob es sich wirklich um eine eigene Erfahrung handelt oder ob sie eventuell von einer Bezugsperson unreflektiert übernommen wurde. Praktisch bedeutet das, immer wieder die Fragen nach dem *Warum?* und dem *Wozu?* zu stellen.

Ausgehend davon bietet die Anforderungsanalyse die Gelegenheit, vor der Modellierung von Anwendungsfällen diesen wichtigen Fragen Raum zu geben. Manchmal wird viel zu schnell das Ziel formuliert. So ist es weit verbreitet, den Gewinn als Unternehmensziel zu betrachten. Aber ist der Gewinn nicht vielmehr eine Bedingung für den Fortbestand des Unternehmens? Das Ziel des Unternehmens leitet sich doch eigentlich aus der Gründungsvision aus dem Inhalt der Arbeit ab. Im Trailer zum Film „Die stille Revolution“ ist zu hören bzw. auf der zugehörigen Internetseite zu lesen: „Wir haben in der Vergangenheit viel *Know-how* gewonnen. Aber wir haben das *Know-why* verloren.“ [StRe18]. Allein das allgegenwärtige Thema Digitalisierung verliert ohne die Antwort auf die Frage *Wozu?* seinen Sinn. Der Focus muss sich also auch auf die Fragen richten: *Warum tun wir etwas?* oder *Wozu?* bzw. *Worin liegt der Sinn?*. Wenn wir etwas tun, nur weil wir es können, genügt dies nicht wirklich als Begründung.

Die Anregung, eigene Haltungen zu ergründen und zu prüfen, kann in den Lehrveranstaltungen auch unterstützt werden durch Hinweis auf die Erfolgsmuster (der Proaktive, der Vielsehende, ...) und Antipattern (zu viel des Guten, Toolistan, ...) von Peter Hruschka und Gernot Starke [HS18]. Ihre These ist: Gute Systeme entstehen durch gute Arbeit der Architekten. Diese gute Arbeit aber basiert neben den Kompetenzen eben auch auf entsprechenden Haltungen, die in den oberen Hirnschichten als Erfahrungen strukturell verankert sind. Neue Erfahrungen führen auch zu neuen Haltungen.

4 Autorität begegnen, Autorität ausüben

Auftretende Konflikte in einer Projektgruppe können zum Thema Autorität führen. Der Autoritätskonflikt ist wohl der erste Konflikt, dem wir Menschen gegenüberstehen. Schlussendlich resultiert er aus unserem Bedürfnis nach eigenem Willen (Autonomie) einerseits und unserem Bedürfnis nach Zugehörigkeit andererseits. Wir alle entwickeln abhängig vom Kontext Strategien, die sich beim Autoritätskonflikt (scheinbar) bewähren und die wir dann quasi als abrufbare Muster im Hirn speichern. Ob uns diese Muster bewusst sind oder nicht, spielt dabei keine Rolle. Nahezu jeder Softwareentwickler begegnet der Situation,

mit Autorität umgehen zu müssen. Im Falle der Übereinstimmung zum Beispiel bei Entwurfsentscheidungen entsteht kein Konflikt, wohl aber, wenn der Konsens abhandenkommt. Welche Möglichkeiten ergeben sich dann für uns, mit diesem Autoritätskonflikt umzugehen? Im Grunde wählen wir als einzelne Person aus nur zwei Möglichkeiten: sich *anpassen* oder dagegen *rebellieren* (mit vielen Graustufen dazwischen). Sowohl das völlige Anpassen als auch das Rebellieren führen über kurz oder lang in der jeweiligen Person zu Inkohärenz, die sich als Stress zeigt. Abb. 2 verdeutlicht dies. Die Projektgruppe verfügt aber noch

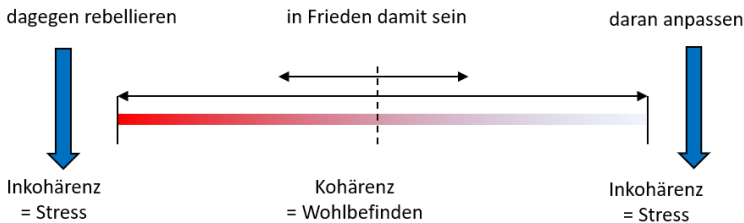


Abb. 2: Mögliche Reaktionen in einem Autoritätskonflikt.

über eine dritte Möglichkeit. Diese dritte Variante stellt Jürgen Pasch [Pa94] in seinem schon etwas älteren Buch vor als den dialogischen Softwareentwurf bzw. das dialogische Prinzip in der Projektgestaltung. Auf diese Weise entstehen nachhaltige Softwaresysteme. Vertreter der modernen Hirnforschung, allen voran Gerald Hüther, bezeichnen dies als Potentialentfaltung [Pot18]. Ist in einem Dialog ein nachhaltiger Entwurf entstanden, dann profitieren das Softwaresystem und die Projektgruppe gleichermaßen davon: die Qualität des Entwurfes ist hoch und die Projektmitglieder erreichen Kohärenz zwischen dem, was sie wollen und dem was sie tun. Schlichtweg fühlen sie sich jenseits von Stress wohl bei ihrer Arbeit.

Inkohärenzen entstehen sicher nicht nur, wenn es um Entwürfe geht. Wie heilsam wäre es, wenn wir Menschen unsere Handlungsmuster in Bezug auf den Umgang mit Autorität und in Bezug auf das Ausüben von Autorität reflektieren und dann ggf. verändern könnten zum Nutzen aller. Unsere Hirne sind plastisch; einer strukturellen Veränderung steht also nichts im Weg. [Hü01]

5 Nicht kommunizierte Annahmen im Projekt

In der Diskussion von Entwürfen zeigen sich weitere Analogien: Bei der Beschreibung von Programmierschnittstellen verweisen wir neben Semantik und Syntax auch explizit auf Vor- und Nachbedingungen sowie gegebenenfalls auf Invarianten. Schlussendlich geht es um Annahmen. Aber wie sieht es mit unseren Annahmen aus, die wir in der Projektarbeit treffen? Kommunizieren wir diese Annahmen wirklich? Der gelegentliche Ausspruch „Ach, ich habe gedacht, dass . . . “ weist unmittelbar auf nicht kommunizierte Annahmen. Die Menge von uns umgebenden sogenannten Krisen zeigt wohl deutlich, dass wir auch im

gesellschaftlichen Leben Annahmen getroffen, sie quasi zum Gesetz erhoben haben und demzufolge nicht mehr hinterfragen. Diese Annahmen mögen zu ihrer Entstehungszeit und im Entstehungskontext durchaus berechtigt gewesen sein. Heute treffen sie vielleicht nicht mehr zu. Die Annahmen „Konkurrenz ist der tragende Entwicklungsmotor.“, „Zeit ist Geld.“ und „Märkte sind Mechanismen.“ scheinen wie fest verdrahtet in unseren Köpfen. Aber wohin haben uns diese und andere Annahmen geführt? Tom DeMarco antwortet auf die Frage: „Haben uns verirrt, kommen aber gut voran.“ [De01](S.43, Kapitel II). Wo bzw. was ist der Kompass, der uns aus dieser Situation herausführen kann? Gerald Hüther sieht in der Besinnung auf unsere *Würde* eine gute Möglichkeit dazu. [Hü18] Tom DeMarco [De01] und Gerald Hüther [Hü97], [Hü20] stimmen darin überein, dass die Kultur der Angst als essentielles Entwicklungshemmnis wirkt. Dies gilt offensichtlich sowohl für Softwareprojekte als auch für unsere Gesellschaft. Im Rahmen der Projektarbeit taucht häufig eine weitere nicht explizit kommunizierte Annahme auf: die Projektleitung habe die Aufgabe, Projektmitarbeiter zu motivieren. Nach meiner Erfahrung und Beobachtung gibt es keine extrinsische Motivation. Vielmehr schafft das, was wir extrinsische Motivation nennen, Abhängigkeit von Geld, von Noten, von Lob und manchem mehr. Allerdings können wir eine Arbeitskultur ohne Angst schaffen, die der (intrinsischen) Motivation der Projektmitarbeiter Raum bietet, eine Kultur, in der sich Potentiale entfalten können. Diese Kultur baut auf Wertschätzung, auf Achtung, ist verbunden mit Achtsamkeit und Gewahrsein und führt unmittelbar zu der Frage *Was ist eigentlich Arbeit?*. Arbeit wurde im Laufe der Zeit immer mehr zum Gelderwerb, vielleicht ist sie aber eher soziale Teilhabe.

Wenn wir Softwareentwicklung im Kontext von Programmierparadigmen diskutieren, dann wird deutlich, dass diese damals neuen Paradigmen die Antwort auf spezielle Rahmenbedingungen waren. Genau genommen haben die Eltern der Objektorientierung Softwaresysteme damals quasi „neu gedacht“. Das führte zu erheblichen Vorteilen ebenso auch zu Konsequenzen. Wenn wir die uns umgebende Wirklichkeit betrachten, dann drängt sich der Gedanke auf: Sollten wir nicht auch in diesem Kontext *neu denken* [Gö21]? Neue Denkweisen schaffen Raum, um Probleme zu lösen, die durch bisherige Denkweisen entstanden sind. Albert Einstein gab uns diese Erkenntnis schon an die Hand. Neue Denkweisen finden wir aber nur, wenn wir veraltete, inzwischen hinderliche Denkweisen als solche erkennen und konsequent über Bord werfen. Und diese veralteten Denkweisen sind meist verbunden mit alten, nicht mehr in Frage gestellten Annahmen. Der gedankliche Weg von nicht kommunizierten Annahmen sowohl in der Anforderungsanalyse als auch im Softwareentwurf zu nicht (mehr) kommunizierten Annahmen in unserer Gesellschaft ist ein kurzer. Andreas Reckwitz diskutiert die Frage, ob digitale Technologien auch eine Kulturalisierung des Technologischen und des Sozialen betreiben. Um auf die Grenzen der These, wir befänden uns in einer Informations- und Wissensgesellschaft, hinzuweisen, unterscheidet er nicht nur Daten und Informationen, sondern auch Kulturformate: „Das Internet ist zu erheblichen Teilen eine Affektmaschine. Seine zirkulierenden Bestandteile erregen, unterhalten, stimmen freudig, entspannen, hetzen auf oder bewirken, dass man sich angenehm aufgehoben fühlt.“ ([Re17], S.234f.) Wäre es nicht notwendig, diese Denkanstöße

aus ethischer Perspektive insbesondere auch mit Studierenden der (Medien-)Informatik zu diskutieren?

6 Reflexion als wichtige Methode zur Überprüfung

Reflektieren, das Innehalten und Prüfen, ist wichtiger Bestandteil der Softwareentwicklung. Bewusst und regelmäßig in kurzen Abständen angewandt, trägt Reflexion agiles Arbeiten und bietet Verbesserungsmöglichkeiten insbesondere auch für Softwareentwürfe. So wie beim Entwickeln und Weiterentwickeln eines Softwaresystems durchaus auch Fehler entstehen können (technische Schulden), so kommt es zu fachlichen Schulden bei mangelnder Anforderungsanalyse. Durch bewusstes Innehalten und Anwendung von geeigneten Analysewerkzeugen sind wir in der Lage, diese Schulden zu erkennen und darauf zu reagieren, so wie es Carola Lilienthal für Softwareentwürfe empfiehlt [Li16]. Das Analysieren und Verbessern setzt aber auch die entsprechenden Haltungen voraus; das äußert sich z.B. in der Planung, in konsequenter Durchführung, in einem Qualitätsbewusstsein.

Vielleicht kommt es analog auch zu ungewollten gesellschaftlichen Schulden, die nach Korrektur verlangen. Aus welcher Perspektive wir Softwareentwicklung bzw. unsere Gesellschaft auch betrachten, schlussendlich geht es auch darum, dass wir uns die Rahmenbedingungen und die Konsequenzen unseres Tuns in kleinen Schritten bewusst machen. Dabei spielt die Reflexion eine zentrale Rolle; die Reflexion wird zum Rhythmus der Arbeit, sie sollte auch zum Rhythmus des persönlichen und des gesellschaftlichen Lebens überhaupt werden. Aber Reflexion braucht Zeit und Raum; der Effizienzwahn der vergangenen Jahrzehnte hat mit dem Beseitigen der Spielräume diese wichtigen Voraussetzungen für reflektiertes Handeln in manchen Softwareprojekten und ebenso in anderen Bereichen verdrängt. [De01]

7 Fazit

Es ist wohl kaum mehr zu übersehen, dass wir als Gesellschaft vor disruptiven Veränderungen stehen. Fragen wie: *Wie wollen wir eigentlich leben und arbeiten?*, *Was eigentlich ist Arbeit?*, *Wer wollen wir sein?* drängen sich auf. Softwareentwickler stehen mit ihrer Arbeit mitten in diesem Veränderungsprozess, vielleicht sogar an einer zentralen Stelle. In den Lehrveranstaltungen zum Software Engineering über technische Themen hinaus zu gehen, soziale und ethische wie auch ökologische Aspekte einzuschließen, bedeutet, die Studierenden einzuladen, der Ohnmacht die Bewusstheit entgegenzusetzen, um gemeinsam einen Weg in eine neue Gesellschaft zu finden, in eine Gemeinschaft, von der Maja Göpel behauptet „Wir können auch anders“ [Gö22]. Vielleicht entsteht diese neue Welt nicht mehr nur vordergründig durch Bearbeitung oder Verarbeitung (Ressourcenausnutzung), sondern durch Erarbeitung [MH20]. Wir alle sind Suchende auf diesem Weg. Auch die heutigen Studierenden werden es sein, die diese Zukunft maßgeblich gestalten und das auf der Basis ihrer Haltungen und damit auf der Basis ihrer heutigen Erfahrungen. Als Lehrende können

wir sie einladen, über den Tellerrand hinaus zu schauen, interdisziplinär zu denken und zu handeln und dabei neue Erfahrungen zu sammeln. Wir können sie mit unseren Fragen ein Stück des Wegs begleiten auch oder gerade im Software Engineering.

Literatur

- [De01] DeMarco, T.: Spielräume - Projektmanagement jenseits von Burnout, Stress und Effizienzwahn. Verlag Hanser, 2001.
- [Gö21] Göpel, M.: Unsere Welt neu denken - Eine Einladung. Verlag Ullstein, 2021.
- [Gö22] Göpel, M.: Wir können auch anders - Aufbruch in die Welt von morgen. Verlag Ullstein, 2022.
- [HS18] Hruschka, P.; Starke, G.: Knigge für Softwarearchitekten. Verlag entwickler.press, 2018.
- [Hü01] Hüther, G.: Bedienungsanleitung für ein menschliches Gehirn. Verlag Vandenhoeck & Ruprecht, 2001.
- [Hü18] Hüther, G.: Würde Was uns stark macht - als Einzelne und als Gesellschaft. Verlag Knaus, 2018.
- [Hü20] Hüther, G.: Wege aus der Angst - Über die Kunst, die Unvorhersehbarkeit des Lebens anzunehmen. Verlag Vandenhoeck & Ruprecht, 2020.
- [Hü97] Hüther, G.: Biologie der Angst - Wie aus Stress Gefühle werden. Verlag Vandenhoeck & Ruprecht, 1997.
- [Li16] Lilienthal, C.: Langlebige Softwarearchitekturen - Technische Schulden analysieren, begrenzen und abbauen. Verlag dpunkt, 2016.
- [MH20] Metelmann, J.; (Hg.), H. W.: Imagineering - Wie Zukunft gemacht wird. Verlag Fischer, 2020.
- [Pa94] Pasch, J.: Softwareentwicklung im Team – mehr Qualität durch das dialogische Prinzip bei der Projektarbeit. Verlag Springer, 1994.
- [Pot18] Akademie für Potentialentfaltung: Gemeinsam über sich hinauswachsen, 2018, URL: <https://akademie fuer potential entfaltung.org/>, Stand: 10.01.2023.
- [Re17] Reckwitz, A.: Die Gesellschaft der Singularitäten - Zum Strukturwandel der Moderne. Verlag Suhrkamp, 2017.
- [StRe18] Kristian Gründling and Bodo Janssen: Die stille Revolution, 2018, URL: <https://www.die-stille-revolution.de/#vision>, Stand: 06.01.2023.
- [TG20] Teobald Werner, Gerald Hüther: Eine Frage der Haltung!, 2020, URL: https://www.gerald-huether.de/wp-content/uploads/2020/06/Eine_Frage_der_Haltung.pdf, Stand: 06.01.2023.

Session 2

Teaching Object-Oriented Programming with the Objects-first Approach: An Experience Report

Sandro Speth¹

Abstract: Lecturers worldwide continue to discuss suitable approaches to teach students programming concepts in general and object-oriented programming in particular. When using the so-called objects-first approach – in contrast to the established bottom-up approach – students start right from the beginning with objects and their interfaces and then only gradually learn more object-internal concepts, such as implementing operation bodies by using variables and control flow constructs. This gives students, from the start, a better object-oriented way of thinking, which is often much harder to learn properly in bottom-up approaches. In this paper, we describe the structure and design considerations and report our experience in implementing our introductory programming course based on the objects-first approach. We outline which concepts we teach, when and to which extent, and discuss the pros and cons of our implementation, considering student feedback from teaching evaluations.

Keywords: Programming Introductory Course; Objects-first; Teaching Programming

1 Introduction

Over the last decades, lecturers have regularly discussed different approaches to how students can best learn object-oriented programming concepts. The most established approaches are the bottom-up method, where students traditionally learn procedural programming first and then get into object-oriented (OO) concepts, and the objects-first approach [Me09]. In objects-first, often called outside-in, students are initially given only an API with a limited number of features provided by, for example, a mini programming world (MPW). This allows a significant amount of programming language complexity to be abstracted away from students. Typical examples of such MPWs are Kara the ladybug² or the hamster simulator [BB04]. At first, students only work by calling the API of a handful of objects, gradually being taught more and more concepts and thus opening up more elements of the MPW and being able to implement them themselves. This is to help students learn and practice library (re-)use and OO concepts and ways of thinking from the very beginning before going into deeper concepts such as control flow. While objects-first, in comparison to the bottom-up approach, often yields good results for students with no prior experience, students who have programmed previously criticize the approach and claim to be more bored in the early part of the lecture. Additionally, instructors often debate how much content can and should be covered in such foundational programming lectures.

¹ University of Stuttgart, Institute of Software Engineering, Universitätsstraße 38, 70569 Stuttgart, Germany
sandro.speth@iste.uni-stuttgart.de

² <https://www.swisseduc.ch/informatik/karatojava>

For this reason, we describe in this paper our experiences with the objects-first approach in our programming introductory course at the University of Stuttgart. We describe the content we cover in our lecture, its depth and sequence. For our course, we implemented an OO variant³ of the hamster simulator in Java which we explain briefly. Finally, we discuss the presented experience, considering the students' evaluation at the end of past iterations. This paper is intended to guide other instructors on how they might structure their programming foundations using the objects-first approach and our tooling.

In Abschnitt 2, we describe the size of our course and how it is organized. Abschnitt 3 explains briefly our used MPW. In Abschnitt 4, we discuss which concepts we teach and their sequence. Finally, we discuss our experience in Abschnitt 5, related work in Abschnitt 6, and conclude in Abschnitt 7.

2 Course Structure

Our course consists of two weekly lectures of 90 minutes, one weekly tutorial of 90 minutes (4+2 model), and one lecture hall exercise of 90 minutes. The course is worth 9 ECTS points, which amounts to a total effort of 270 hours for the students. The lecture takes place every winter term, and more than 800 students from more than 20 study programs are enrolled. Usually, the students are studying in their first semester, hence, without prior experience in programming besides their personal background. At the end, the students can evaluate the course (c.f., Abschnitt 5) and have to pass a written exam. This report is based on experience of four iterations of this course. In previous work, we have discussed a detailed description of the structural set-up during the Covid-19 pandemic [Sp22]. For a practical in-depth study of the content learned, students receive a weekly exercise sheet, which consists of an attendance part and a homework part. The students can work on the attendance part in groups of 2 or 3 onsite. Programming tasks are to be solved and discussed using pair programming. In addition, students can discuss and seek help from the tutor during the exercise. The homework part is to be solved independently in the same groups of two or three students and thematically relates to the topics of the previous week's attendance assignments. Each exercise sheet usually includes both programming tasks and theory tasks where students are expected to discuss topics with each other and the tutor to practice correct terminology. Furthermore, for student teachers, most exercise sheets contain additional exercises to discuss didactical questions [BBF20]. In addition, in the lecture hall exercise, the contents of the lecture are repeated, and questions from the students are discussed.

3 Objects-first with Mini Programming Worlds

Mini programming worlds provide a small, concise API that students can use without being directly overwhelmed. With the given API, students implement several small scenarios,

³ <https://github.com/SQAHamster/plain-java-hamster>



Abb. 1: Overview of the topic sections in our lecture

which allow them to learn different concepts of OOP step by step. Especially in the objects-first approach, this offers the advantage of abstracting a lot of language-specific boilerplate code and complex concepts at the beginning and focusing on objects and their attributes as well as interaction between objects. However, many existing MPWs have limited support for a fully modern object-oriented API. For this reason, we have implemented a new object-oriented simulator in Java which is conceptionally based on the original hamster simulator by Boles et al. [BB04]. Objects in our simulator are, e.g., hamsters, grains, walls, and tiles on which the hamsters move and the walls lie. The hamsters can move one tile ahead, turn left, pick up, and put down grains, check if the tile in front of them is clear and check if there is a grain available on the current tile. Based on this, many scenarios of varying complexity can be programmed, such as solving a maze. Furthermore, OO concepts like inheritance can be integrated into the world, for example, to make the hamsters more powerful.

4 Content of the Lecture

In this section, we describe the content we cover in our lecture and its sequence and depth. In total, our lecture consists of 21 content lecture units, each of which goes over one or two lecture slots. The lecture units are structured in four topics sections as depicted in Abb. 1. In general, we discuss associated style guides for all concepts from Meyer's „Touch of Class“ [Me09] and other clean code rules.

Object & Class Foundations and Usage: (1) In the beginning, we discuss the motivation and introduction to computer science with the students. We then introduce elementary computer science concepts such as compilers and discuss the importance of software quality. (2) Then, we introduce students to the hamster simulator and form a basic understanding of programs. For this, we show simple examples of hamster games to demonstrate how programs can look like. We also decompose and format initial programs and call operations on a hamster object. In particular, we define objects and entities, discuss the difference between commands and queries and why a clear separation is important. We motivate object orientation for students by building stories through sequences of interactions between objects and their proximity to the real world. (3) In the second week, we discuss the structure of programs, addressing instructions and expressions, and defining lexis, syntax, semantics, and pragmatics of programs. In the process, students learn the steps of a compiler and build simplified abstract syntax trees for classes with operations only. In addition, we discuss static semantics rules. One objective of this lecture is to identify errors in program code and correctly classify them into error classes. (4) We then discuss the different types

of interfaces and their documentation. In particular, we focus on high-quality JavaDoc comments. Furthermore, we teach the concept of classes, as well as how to use classes and explain namespaces using packages as an example. A relevant part of this lecture unit is contracts, preconditions and postconditions of operations, and class invariants. (5) The fifth lecture unit deals with foundational logic.

Data Types, Variables, and Control Flow: (6) After learning the basics of using objects, we look at object creation, addressing object references and null and this references and their semantics. In addition, we look at the concepts of allocating, initializing, and assigning, explain constructors, and discuss the Java Optionals API. (7) To program more complex scenarios, we then teach control flow structures. In doing so, we begin with the concept of an algorithm and start with sequences. Then, students learn branches, the correctness of branches, and nested branches. For loops, we first teach the three normal types of loops without the foreach loop, dive into the correctness of loops via loop variants and invariants, and finally cover error handling. (8) In the eighth lecture unit, we discuss types and variables, as well as visibility scopes, releasing variables, and read-only variables. We also delve into primitive data types and discuss literals and type conversion. Furthermore, we discuss non-primitive data types and the difference between object equality (equals) and reference equality (same). Finally, we discuss how to make objects immutable.

Own Java Classes and Inheritance: (9) Lecture unit nine deals with programming own Java classes. We discuss static and non-static elements, functional decomposition, the uniform access principle, and access control via visibilities. Furthermore, we look at offensive and defensive programming and when to choose which variant. (10) After students know all the basic programming concepts, we discuss object-oriented inheritance and polymorphism. In doing so, we begin with the basics of inheritance and discuss the module viewpoint and type viewpoint. Afterwards, we focus on abstraction and interfaces, as well as overloading and overriding operations. Additionally, students will learn about polymorphism, static and dynamic types, and dynamic binding. Furthermore, type conformance, constructor chaining, access to operations and attributes of the parent class, and multiple inheritance are discussed. Finally, to ensure the correctness of inheritance, we explain Liskov's substitution principle to the students. (11) Following, we discuss concepts and definitions of various data structures, classify them into a taxonomy, and teach the foreach loop and iterators. We also go into more detail about the proper use of special collections and the map, and explain why arrays are usually to be avoided. (12) At this point, 2/3 of the lecture period is over, and the Christmas lecture roughly outlines various programming languages.

Software Engineering Basics: (13) We begin the last third with a detailed discussion of clean code. (14) In addition, we discuss recursion with examples and go into recursive data structures, the correctness of recursion, and recursion elimination. (15) In the fifteenth lecture unit, we deal with modelling. Thereby, we introduce UML and particularly consider object diagrams, class diagrams, the relationship between class diagrams and implementation, as well as sequence diagrams. (16) Subsequently, we give a brief insight into various software engineering topics, such as development processes, requirements engineering, and software

architecture. (17) Further, we elaborate on the motivation and fundamentals of testing. Thereby, we discuss the identification of test cases, black box and white box testing, as well as automated (unit) testing. (18 & 19) Towards the end of the semester, we look at passing functions in OO languages, lambdas, inner and anonymous classes, method references, and the Java Streams API. (20 & 21) Finally, we look at program proofs, weakest precondition, and GUIs with events and listeners.

5 Discussion

Of the approximately 800 students who start in the lecture, roughly 550 students take the exam, with approximately 55% failing the exam, which seems a common rate for programming introductory courses in Germany. The exam is written on paper and consists of programming and concept exercises.

Furthermore, we evaluate each iteration of our lectures with a questionnaire that can be voluntarily completed online by the students. Due to the Covid-19 pandemic over the last few years, only approximately 100 to 150 students participated in the survey each time. In the questionnaire, students can rate various questions on a Likert scale from 1 (strongly agree) to 5 (strongly disagree) and provide free text responses for further feedback. There are separate surveys for the lecture, exercises, and lecture exercises. The questions primarily focus on the quality of organization, clarity of content structure, communication of learning objectives, comprehensibility of explanations, motivation, and addressing the students' concerns. The students rate the lecture between 2 and 2.3, the exercises between 1.8 and 2, and the lecture exercise between 1.3 and 1.8. Overall, therefore, students appear to be very satisfied with the course.

In the free-text answers, the students are particularly positive about the comprehensible and clearly well-structured content and the high amount of practical programming tasks with the hamster simulator. However, the students criticize the overall amount of material in the lecture, as well as the effort of the exercise sheets for students without prior knowledge. Likewise, students from technical engineering courses, such as mechanical engineering, would prefer to learn languages such as C++ instead of Java.

Furthermore, it is notable that students without previous knowledge in programming take up the objects-first approach very positively, while students, who had already learned to program with a bottom-up beginning, evaluate the objects-first beginning as negative. One problem with the approach is that students with previous knowledge get bored at the beginning and then stay away from the lecture but do not catch the right moment to join again. In addition, in some topics where students already have prior knowledge, we teach more complex content that students are unfamiliar with and thus miss, e.g., loop invariants or documentation of contracts. Nevertheless, we could not find a clear indication in the exam that having no prior knowledge leads to better results or vice versa. The use of the hamster simulator brings great advantages for students without prior knowledge due to its simplicity

at the beginning. Still, students often wish to use other MPWs later in the semester that allow more complex scenarios. Here, e.g., one could build an MPW that, instead of just having hamsters that can directly pick up grains, could work with raccoons whose worlds are more complex and must meet preconditions to open garbage cans.

In summary, an objects-first approach with MPWs is particularly useful for teaching object-oriented programming to students without prior experience. However, more complex scenarios and MPWs should ideally be enabled as the semester progresses, and multiple programming languages should be supported in the case of diverse courses. As a result, in addition to our hamster simulator, written directly in Java, we have built MPWs using MDSD frameworks [Fu21, FB21]. Currently, these can generate a hamster simulator in C++ and Java.

6 Related Work

Teachers often use the education IDEs BlueJ [Kö03, BKG06] or Greenfoot [Kö10, HK04] in their Java-based OO programming courses, e.g., [SZ07, GM08]. Both IDEs allow students to instantiate objects via a class diagram and interact with them graphically, thus offering a native objects-first approach. Additionally, both editors contain a source code editor with basic editing options but good visualization of block scopes. In our first three iterations, we used BlueJ in our first few exercise sheets before switching to industry-ready IDEs. However, BlueJ and Greenfoot currently do not run with newer Java versions (later than 11), and the students were fighting bugs in BlueJ on a regular basis. Therefore, we decided to add object manipulation directly to our hamster simulator and directly start with industrial IDEs. One reason to use IDEs such as Eclipse, IntelliJ, and Visual Studio Code as we want to teach our students more elaborate IDE features, such as refactorings, to prepare them for later courses and industry. Furthermore, we decided against them as we plan to support our hamster simulator in languages other than Java.

Schmolitzky et al. [SZ07] presented the structure of their programming introductory courses at the University of Hamburg. In general, they follow a similar approach. However, their course is divided into two semesters and has a different order of topics. For example, they include the implementation of data structures and algorithms in the first semester, which we exclude as there is another course at our university. Furthermore, our course includes basic SE concepts instead of focusing on programming only. Cooper et al. [CDP03] outlined their approach to teach objects-first with their own 3D MPW platform called Alice. However, their approach only teaches a scratch-like programming language, and they do not state the topics they cover in their course.

7 Conclusions & Future Work

In this paper, we described the structure of our programming introductory lecture and our experiences with the objects-first approach. In particular, we addressed the order and depth of various topics covered. Finally, taking into account student feedback from course evaluations over the past few years, we discussed when and how to teach programming using the objects-first approach. This paper intends to help other lecturers decide whether teaching according to objects-first is appropriate and what content can be taught within the available time. A major criticism of our implementation of the concept is that some students, especially those with previous experience, find our hamster MPW too boring after some time. For this reason, in the future, we plan to generate different worlds of varying complexity and programming languages through model-driven MPWs [FB21], thereby ensuring a more significant increase in complexity and variety.

Literaturverzeichnis

- [BB04] Boles, Dietrich; Boles, Cornelia: Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell, Jgg. 1. Springer, 2004.
- [BBF20] Becker, Steffen; Bescherer, Christine; Fest, Andreas: Reflective Pedagogical Practice on and in Introduction to Programming and Software Engineering. In: Tagungsband des 17. Workshops SSoftware Engineering im Unterricht der Hochschulen"2020. Jgg. 2531 in CEUR Workshop Proceedings. CEUR-WS.org, S. 7–10, 2020.
- [BKG06] Barnes, David John; Kölling, Michael; Gosling, James: Objects First with Java: A practical introduction using BlueJ. Pearson/Prentice Hall, 2006.
- [CDP03] Cooper, Stephen; Dann, Wanda; Pausch, Randy: Teaching objects-first in introductory computer science. *Acm Sigcse Bulletin*, 35(1):191–195, 2003.
- [FB21] Fuksa, Mario; Becker, Steffen: Mini Programming Worlds: Teaching MDSD via the Hamster Simulator. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). IEEE, S. 696–701, 2021.
- [Fu21] Fuksa, Mario: Redesigning the Hamster Simulation. Masterarbeit, University of Stuttgart, 2021.
- [GM08] Gallant, Randy J; Mahmoud, Qusay H: Using Greenfoot and a Moon Scenario to teach Java programming in CS1. In: Proceedings of the 46th Annual Southeast Regional Conference on XX. S. 118–121, 2008.
- [HK04] Henriksen, Poul; Kölling, Michael: Greenfoot: combining object visualisation with interaction. In: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. S. 73–82, 2004.
- [Kö03] Kölling, Michael; Quig, Bruce; Patterson, Andrew; Rosenberg, John: The BlueJ System and its Pedagogy. *Computer Science Education*, 13(4):249–268, 2003.
- [Kö10] Kölling, Michael: The Greenfoot Programming Environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–21, 2010.

- [Me09] Meyer, Bertrand: Touch of Class, Jgg. 51. Springer, 2009.
- [Sp22] Speth, Sandro; Krieger, Niklas; Reißner, Georg; Becker, Steffen: Teaching during the Covid-19 Pandemic — Online Programming Education. In: Software Engineering im Unterricht der Hochschulen (SEUH 2022). Gesellschaft für Informatik, S. 89–94, Februar 2022.
- [SZ07] Schmolitzky, Axel; Züllighoven, Heinz: Einführung in die Softwareentwicklung-Softwaretechnik trotz Objektorientierung? In: SEUH. S. 87–100, 2007.

Teaching Cyber-Physical Systems in Student Project Groups: How Do Alumni Assess the Experience in Retrospective?

Henning Schlender¹³, Ralf Stemmer¹³, Kim Grüttner¹³, Günter Ehmen¹³, Bernd Westphal¹³,
Friederike Bruns²³

Abstract: The development of Cyber-Physical Systems (CPS) requires software engineering competences in an interdisciplinary context including physics, mechatronics, and electrical engineering. In addition, social skills and teamwork competences are necessary since the current complexity of CPS exceeds the capacity of a single human. The Computer Science faculty at the University of Oldenburg offers project groups (PGs) in the Master's curriculum to particularly strengthen these competences. To this end, small teams of students work jointly during two semesters on large software or systems engineering problems, which are often inspired by research projects with industrial partners.

In this paper we report on a survey that we conducted among alumni of a set of nine PGs that worked on different challenging CPS problems. The main questions were how alumni who already started their professional career assess the learning experience and different aspects of the design of the CPS PGs in hindsight. The responses indicate a good learning success on the main learning objectives of the considered CPS PGs, and a large majority of respondents considers the course to be comparably relevant or very relevant in the overall Master's curriculum.

Keywords: Software Engineering, Systems Engineering, Cyber-Physical Systems, Project Group, Alumni-Survey

1 Introduction

Cyber-Physical Systems (CPS) are already omnipresent in our daily lives and are still increasingly entrusted with safety-critical tasks in domains such as transportation systems, medical devices, or factory automation. Among the characteristics of CPS are the aspects that they need to interact with a physical environment in a reliable, timely, and safe manner. Hence, the development of CPS requires competences from many different disciplines including physics, mechatronics, electrical engineering, and of course computer science. The latter contributes competences from areas such as embedded HW/SW design, real-time systems, control theory, and software and systems engineering methods that deal with the high quality demands, the particular care that is needed for functional and extra-functional properties, as well as mixed criticality components and Safety Integrity Levels (SIL). Last

¹ German Aerospace Center, Institute of Systems Engineering for Future Mobility, Oldenburg, Germany

² Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany

³ henning.schlender@dlr.de, ralf.stemmer@dlr.de, kim.gruettner@dlr.de, guenter.ehmen@dlr.de,
bernd.westphal@dlr.de, friederike.bruns@uol.de

but not least, CPS development needs to be economic in the sense that the necessary quality is achieved with bounded time, effort, and costs.

Education for research in methods and tools for CPS as well as for the development of CPS is correspondingly challenging in a traditional curriculum which only relies on instruments like lectures, exercises, seminars, individual projects, and single term practicals. Each of the above areas of knowledge and competence can easily define its own module, but time constraints usually do not allow adequate application of the techniques to more than rather small toy examples. In addition, there is the challenge to address one aspect that is particularly relevant to CPS development: That CPS development is not only about technology but about humans [LL07]. Today's CPS have reached a complexity level that by far exceeds the intellectual capacity of a single human engineer thus CPS development requires particularly strong soft skills such as social interaction, written and oral communication, and adequate attitudes towards responsibility. The Computer Science faculty at the University of Oldenburg has a long tradition of approaching many of these challenges with an instrument called *project group* (PG) in the graduate studies (optional since the 1990s, compulsory with the introduction of the Master's degree). A PG is a module that runs for two semesters (one year) with a workload of 24 ECTS (or 16 hours per week). The enrolled 6-12 students stay together for the whole duration and complete a project in a team. The team starts from a given, relatively open problem definition, which needs to be detailed by the students for a complete development cycle up to the complete realisation of the solution. Similar modules are offered, e.g., at TU Dortmund or the Universities at Bonn and Paderborn.

In this paper, we consider a particular PG design with the purpose to address the above mentioned educational challenges of the domain of Cyber-Physical Systems. We report on an effort to investigate one of the most relevant questions in teaching: How well does the module meet its challenges? Do students learn a lot and do they acquire the right knowledge and competences? As it is notoriously difficult for students to help in an assessment of these questions *during* their studies, we have taken the effort to conduct a survey among students who have already completed their studies and started their professional career. To obtain an as large as possible data set, we approached participants from nine completed CPS PGs and invited them for a questionnaire on how they look back on their PG experience.

2 Related Work

Marwedel et al. and discuss different curricula in the direction of CPS [Ma20]. The authors talk about the challenges in the CPS area and an important one was the interdisciplinary education. The authors also see a problem in the usually limited resources in terms of personnel, time and often also financial means. It also reads that teamwork is the key to bridging the different disciplines needed in the CPS field. However, long-term team-based projects were only marginally mentioned and not sufficiently discussed.

In [BS12] a 12 week course about CPS design is described. The authors write, that CPS design requires a broad field of knowledge and often there is not enough time to teach all aspects in detail. In [Go15] a 17 week project-based learning approach is used. The authors support the claim, multi-disciplinary projects can help to learn optimization across different engineering disciplines. [LSJ12] reports about capstone projects, which are 2 semester single student projects. They mention the impact on the career trajectory of students. In [TH16] the authors observed that interpersonal skills like working in cross-disciplinary teams are important in the CPS domain. They also believe that capstone projects are an important part of CPS education, which is a bit contradictory as they are handled by individuals. Student projects in team size 4 are described in [Li13]. The project covers the steps of requirements analysis, implementation and testing. [Sc11] describes one semester student group tasks with a size of 8 students. The paper places particular emphasis on the importance of group work. Our PGs often have a team size of 12 people and runs for two semesters. This allows us to pose a much more complex problem formulation. In addition, the size of the group means that there is a much greater need for coordination. Capstone projects, on the other hand, cover relatively complex problems but do not include the very important aspect of teamwork. Also in [Wa15] the challenge of interdisciplinary education is mentioned. The authors describe their approach to implement a 4 course approach on education in CPS. The courses cover the complete life cycle of a system and it also seems to be a complex system. However, the project theme seems to be the same in each cycle. The topics of our project groups are always different.

In [St20] a survey is done about a project-based course design afterwards. The course design was well accepted and the students found it very interesting. Repetitive interviews and questionnaires are used in [Ri17] for evaluation of a 16 week model-driven engineering CPS project class. In [Be22] a survey is done to evaluate a practical oriented one semester classroom and lab approach. None of the surveys mentioned above includes the alumni perspective and thus the effects of such projects on the later job.

3 Project Groups for Cyber-Physical System Education

The PG module in the Master's program at the University of Oldenburg in general has two major learning objectives. Firstly, to strengthen competences related to teamwork (cf. [Ma20]), including (self-)organisation and management of teams, resolving conflicts, taking responsibilities, and information and document management. Secondly, to provide an environment where students can experience larger problems than what can be handled by tiny teams of 1 to 3 persons. This includes the challenge to deal with imprecisely stated, hardly manageable requirements, the possible need to develop partial solutions, and the need to establish appropriate quality assurance measures.

In this paper, we consider a particular design of such PGs that includes particular challenges from the domain of Cyber-Physical Systems. The most obvious particularity of this PG design is the consideration of (special purpose) hardware and its interaction with a physical

environment together with the necessary software development. Fig. 1 gives brief outlines of some concrete instances of the PG design reported on here; the experience of PG ‘AA’, for example, even turned into a publication [Sc15] which gives many more technical details. From the descriptions, we see that the considered problems have been designed to include embedded hardware and aspects of control theory, hard real-time constraints, and mission or safety-criticality. Such problems allow us to address a number of learning objectives that are specific to the domain of Cyber-Physical Systems. Firstly, to include budget planning in the project management (students need to buy parts of the equipment from an available budget) as well as time management for resources (there needs to be a schedule of who is using the prototype hardware for testing when). Secondly, to deal with interdisciplinary projects and address particular challenges to the organisation of the work (PG participants need to specialise to solve the problem but have to be sufficiently familiar with other specialisation areas to integrate the partial solutions; also, CPS PGs need to use a much more heterogeneous set of development tools than pure software PGs). There are also higher demands wrt. thorough risk-analysis and management. It is, for example, a risk that hardware is not delivered in time, that electronic components are harmed by electrostatic discharge effects, or that the whole system is damaged in trial runs. Overall, students should experience the importance of well-defined roles (at least project manager, test manager, and document manager) and the importance of thorough requirements analysis, component specifications, and quality assurance measures (errors in these activities can usually not just be work-around-ed in software in CPS projects).

From the teaching perspective, there are two main challenges towards reaching these learning objectives. One is not to over- or under-strain the students: At the beginning of the project it must be clear that the problem (or a sufficiently large sub-problem) can be solved with the given amount of work (6-12 students, 24 ECTS each) and taking into account the uncertainties mentioned above. The second one is fair marking/grading since students need to obtain individual marks/grades by the study regulations. The first challenge is addressed on different levels. Problems for CPS PGs are usually inspired by subjects of active or completed joint research projects including the University of Oldenburg, research institutes, and partners from the industry. Most of the PGs considered here were carried out in close corporation with the research institute OFFIS e.V.¹ This approach ensured the availability of domain experts who were able to conduct a preliminary analysis and develop a sketch of a “shadow plan” before offering a PG to the students. Integrating members of the research institute as supervisors (and in some cases industry representatives as external mentors) was perceived very valuable since all nine PGs from Fig. 1 completed in time with impressive results. Individual conversation during and directly after the PGs indicate that students understand that they are working on scaled-down variants of real, industrial CPS problems, which, together with the creation of a real, tangible product and the self-determined work in the project, have a very positive effect on the motivation, the enthusiasm, and creativity. Consulting the “shadow plan” helps in assessing the progress of the PG and can be a basis to

¹ In 2022 the transportation department of OFFIS e.V. transferred to German Aerospace Center, Institute of Systems Engineering for Future Mobility.

<p>SmartSystems – 4Wheeler (2004/05). This PG developed a 4-wheeled, remote-controlled robot capable of mapping. They used several sensors (ultra sonic, camera, laser), and combined FPGA implementations and real-time algorithms.</p>	<p>MHA2 (2005/06). This PG developed an FPGA-based mobile PDA capable of running a special hearing aid development software (MHA: Master Hearing Aid). The students developed their own FPGA circuit board and implemented several audio filter functions in hardware.</p>
<p>Eye Fly (2007/08). This PG developed a steady camera system inside of a rolling ball. Part of this project was to merge 12 camera images via image processing to one all-round view.</p>	<p>Micro Urban Challenge (2008/09). This PG developed a small scale autonomous car using a simulation-based development approach. Highlight of this project was stereo-camera based traffic light and traffic sign detection system.</p>
<p>Mobile Solar Power Plant (2012/13). The PG Mobile Solar Power Plant developed a solar driven power plant involving either a Stirling motor or a solar cell. The system included an automatic sun position adjustment mechanism.</p>	<p>AA (2014/15). The PG <i>Avionic Architecture</i> specified, implemented, and tested a mixed-criticality multi-rotor system. The task was to integrate a safety-critical flight control algorithm with a mission-critical payload processing on a single chip. The topic was inspired by an industrial use-case in the research project CONTREX (https://contrex.offis.de).</p>
<p>ViDAs (2009/10). The PG <i>Virtual Driver Assistance</i> developed a driver assistance system for use in simulation environments with a model-based approach using Simulink. The functionality included a lane change assistant, traffic sign recognition and an adaptive cruise control which considered the detected traffic signs. The students also had to develop sensor models based on real sensors.</p>	<p>EmBraAC (2018/19). The PG <i>Emergency Braking Assistant for Fully Autonomous Cars</i> developed a safe evading manoeuvre for an autonomous vehicle based on the FITENTH car with guaranteed execution times. The students modelled the system with contracts to ensure correct timing requirements and verified their system through measurements later (correct-by-construction method).</p>
<p>Guardian (2016/17). This PG implemented a satellite control system on four Xilinx Zynq-7000 FPGA boards and simulated an asteroid interception mission in a simulation called Kerbal Space Program. The students also had to develop a fault injection methods for their testing phase.</p>	

Fig. 1: Goals and achievements of nine PGs on Cyber-Physical Systems.

decide on when supervisors need to intervene (which was only necessary once so far due to a system dynamics model that turned out to be too optimistic compared to the reality). This pre-planning is not visible to the students. They obtain the problem in an intentionally brief and imprecise form during a few introduction sessions. The students acquire the necessary knowledge to analyse the problem in a seminar phase, and develop a system specification, a project plan (with own milestones) for a fixed deadline within the first months and structure their way of working on their own. Results are presented at a mid-term and a closing event. In addition, there is continuous supervision in the form that a small team of supervisors offers counselling in weekly or bi-weekly project meetings. While the students are supposed to

take all decisions on their own, the supervisors may recommend re-assessments if the plans seem to over- or under-ambitious.

The marking/grading approach is based on a scheme with three parts. A student's final mark is determined by a mark for the overall result, a mark for teamwork, and a mark for individual work. The mark for the overall result is based on the project report. Here, it is in our experience important to communicate to the students that even if the PG problem is not perfectly solved, this can still be a very good PG result if the technical reasons are well understood and documented. Otherwise, students may assume that a non-perfect solution counts as a failure which would cause unnecessary strain. The mark for teamwork is assigned by the supervisors according to a detailed marking scheme (including aspects like presence/absence of complaints, responsible assumption of roles, offered or refused cooperation, etc.). The individual mark is based on the contributions of the student to the project as declared by the student. Interestingly, this step has been remarkably smooth in all nine PGs: the teams were always able to agree on who takes credits for which parts. In addition to informal feedback during the team meetings, there is one mid-term review where each student receives individual feedback; earlier interventions are conducted on demand.

All nine PGs from Fig. 1 were successful measured by the marks and the organisers' impression and were subject course evaluations. The design has also been refined over time, for example, later PGs did not schedule all students' seminar talks into a fixed slot in the beginning of the PG time but scheduled the presentation closer to the point in time where the expertise was needed. Yet, course evaluations usually only take place *within* the study times, so a long standing question was how former participants of CPS PGs perceive the learning experience and success in hindsight, after some time or professional work experience.

4 Alumnis' Assessment in Retrospect: Survey Design

The goal of this study was to collect data on the retrospective view on previous PGs on Cyber-Physical Systems. To this end, we have considered participants of nine already completed PGs from a time period between 2004 and 2019 with 93 participants in total. Of these students, 65 were approached individually using the mechanisms of career-oriented social networks. We sent a contact request first which, if accepted, was followed by an explanatory email including a link to the online survey site. with the questionnaire. The site was online for 4 weeks, participants did not receive any reminder after the first message.

Data was gathered by an online survey without the possibility to identify the particular participants. The questionnaire is structured into five parts (cf. Tab. 1). The first block consists of questions for demographic data (current age, and sex). The second block asks for the context of the current employment (where multiple of the three options 'Research', 'Development', 'Teaching', and 'Other' can be selected) as well as the current branch (to be selected over a list of standard branch names). The third and main block asks for a subjective assessment of the learning experience in the PG wrt. nine competence areas,

- How old are you? [..25/..30/..35/..40/..45/..50/50..] • Sex? [f/m/d/prefer no answer]
- In which context are you currently working?
[Research/Development/Teaching/Other; multiple choice]
- In which branch? [single choice over a list of standard branch names]
- During the course of your PG, how much did you learn wrt. the following topic: Project management (leading projects, approach, . . .), self management (own way of working, time management, . . .), taking responsibility, presentation of results, social interaction, documentation, programming, development of hardware, testing?
Each rated on a six-valued Likert scale from 'nothing' (1) to 'a lot' (5).
- Did you get to know new management tools during your PG time? [y/n]
Conditional (if 'y'): Which? [free text] How often do you use one or more of the previously named tools? [five-valued Likert scale from 'not at all' to 'a lot']
- Same questions as before for "dev. tools" and "technologies (languages, hardware, etc.)".
- How relevant do you consider the project group in comparison to other modules in the Master's studies? [five-valued Likert scale from 'not relevant' to 'very relevant']
- How did you perceive the grading/marking in your PG?
[five-valued Likert scale from 'not fair' to 'very fair']
- What did you particularly like about your PG? [free text]

Tab. 1: Questionnaire design.

namely project management, self management, taking responsibility, presentation of results, social interaction, documentation, programming, development of hardware, and testing. The fourth block asks to name tools and technologies that participants got to know during their PG time, and requests a subjective assessment of how often at least one of the named tools the participant uses at the time of the questionnaire. The fifth block focuses on the didactic aspects of perceived relevance of the instrument of PGs in the curriculum and the perceived fairness of markings. The last question is an open question for particularly liked aspects about the respective PG.

5 Alumnis' Assessment in Retrospect: Survey Results

Of the 65 students that were contacted, 47 completed the form. Hence, the survey achieved a remarkable response rate of over 72 %, even without a reminder message or particular incentives connected to filling in the form, and for some students years after finishing their studies. The distribution of sexes in the group of respondents matches the one in the set of candidates and is about equal to the overall distribution in the Master's programme. There is thus no indication that certain sexes particularly choose or avoid CPS PGs. There were responses from almost all age groups so we can assume that the responses refer to multiple

different PGs. Age below 30 was reported by 15 participants, between 30 and 35 another 8, and over 35 the majority of 22 responses. The exact age was not asked for since the answers should not allow to conclude to the particular PG a respondent referred to.

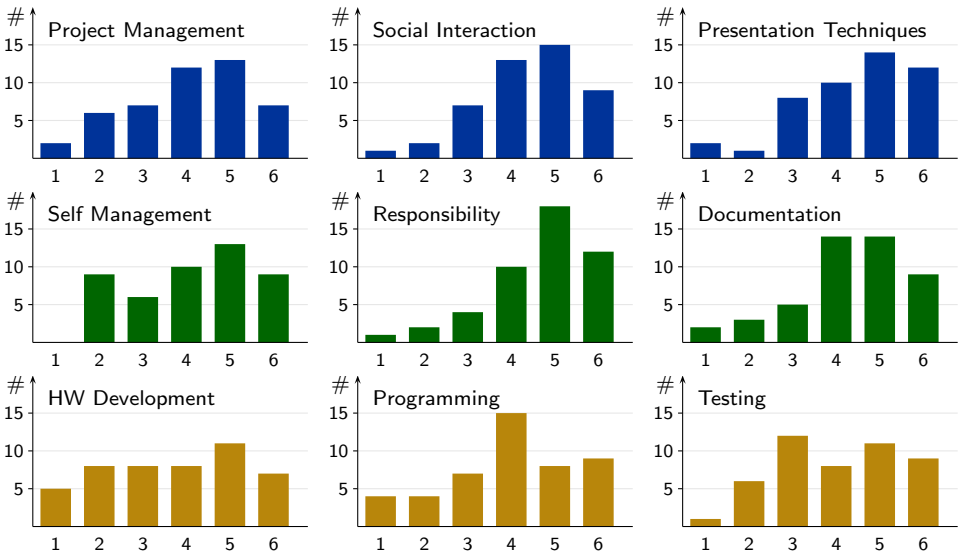


Fig. 2: Subject area learning experience (abs. number of responses, cf. Tab. 1 for the scales 1–6).

Responses regarding the current branch were less informative, probably by a sub-optimal choice of pre-defined branch names. The highest number (29.8 %) is for ‘automotive’, followed with a distance by ‘service’ (8.5 %), ‘transportation’ (6.4 %), and ‘maritime’ (4.3 %). Under ‘other’ (31.9 %), respondents reported ‘energy’ (3) and ‘mechanical engineering’ (3), ‘software’ (2) and ‘robotics’ (2), and one each for ‘university’, ‘navy’, ‘home automation’, ‘agriculture’, and ‘online sales’. The remaining branches were not selected in any response. Regarding the context of the current employment, 16 respondents named ‘Research’, the majority of 30 named ‘Development’, and 5 named ‘Teaching’. Multiple choices were possible, there are 5 overlaps between ‘Research’ and ‘Development’, and 4 overlaps between ‘Development’ and ‘Teaching’. Five participants did not answer this question at all.

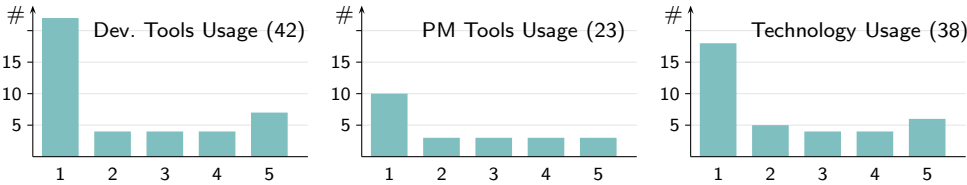


Fig. 3: Usage of tools and technologies met *newly* in PG, hence the number of responses here (in parentheses) is lower than overall questionnaire responses (cf. Tab. 1 for the scales 1–6).

The plots in Fig. 2 give the absolute numbers of responses on the Likert scale for three aspects of learning with three questions each. The first aspect (blue in Fig. 2) is the way of

working in a team, including project management (leading projects, methods, . . .), social interaction, and presentation or communication techniques. The second aspect (green) is the own way of working, including self-management (time management etc.), taking and delegating responsibility, and the documentation of results (for oneself and for team members). And there are the technical aspects (beige), here programming, testing, and (as a particularity of the PGs considered here) hardware development.

In total, 42 respondents (89.4 %) stated that they did get to know new development tools, namely modelling tools (Matlab, Simulink, Stateflow, Scade, CAMELView; 28 mentions overall), chip development tools (Xilinx Vivado, Altera Quartus, etc.; 16 mentions overall), and more generic development tools (LaTeX, make, git, Redmine, etc.; 8 mentions overall). Some project management tools were new to 23 respondents (48.9 %), including Redmine, Jira, MS-Project, Traq, and Slack. Some technologies were new to 38 respondents (80.9 %), including programming languages² (C, C++, Ruby, Python, Assembler; 15 mentions overall), CPS hardware (development) technologies (VHDL, FPGA, System-on-Chip, microcontroller; 13 mentions overall), and specific solutions (CAN, I2C, SPI, etc.; 4 mentions in total). Fig. 3 gives responses to the question for how often at least one of these tools and technologies are used; note that these questions were only presented to those participants who stated that they did get to know a new tool or technology.

Regarding the questions on teaching aspects of the considered PGs, only 7 responses consider the PG (much) less or equally relevant compared to other modules in their study plan (except for the Master's thesis). Only 4 respondents indicated

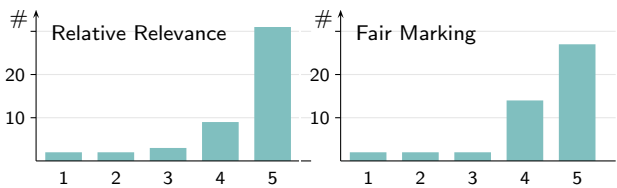


Fig. 4: Relative relevance and fairness of marking. (See Tab. 1)

that they perceived their marking to be not or less fair (cf. Fig. 4). For the free text question on what participants particularly liked, about half of the respondents named the cooperation in the team (40.4 %). A larger number of responses named the practical experience (14.9 %), the large size of the problem worked on in the PG (17.0 %), and the long duration of the module (6.4 %). Other aspects were the dedication of the supervisors (6.4 %), that new topics were addressed (8.5 %), the possibility for a related Master's thesis (4.3 %), and the degree of freedom that the task statement permitted (4.3 %).

6 Discussion

The survey presented in Section 4 is designed to investigate three main questions: Firstly, how do alumni assess the learning success in hindsight, after completion of their studies, and with some experience from a working environment? Secondly, how do alumni assess

² Java is the most taught language at the University of Oldenburg.

the relevance of the teaching instrument of PGs in hindsight and how do they feel about the fairness of their marks/grades? And thirdly, how appropriate is the choice of CPS problems, tools, and technologies for the PGs?

Regarding learning success, Fig. 2 shows that that a large majority of responses leans towards ‘learned a lot’ (Likert values of 4 and above) for project management (68.1 %), social interaction (78.7 %), and presentation techniques (76.6 %). If we consider the values of 5 and above, we still have 42.6 %, 51.1 %, and 55.3 %, respectively. These results indicate that the CPS PGs presented in Section 3 clearly reach the learning objectives in the perception of the responding alumni. Recall that the PGs take place in the graduate studies, hence the students have clearly been exposed to opportunities to practice social interaction and presentation techniques before and still report to have learned a lot in the PG. Whether these results can be attributed to the focus on CPS is unclear since we are not aware of comparable survey data from PGs that do not focus on CPS. By construction (cf. Section 3), CPS PGs clearly provide particular challenges in these aspects.

For the learning aspects responsibility and documentation, the majority is even larger (85.1 % and 78.7 %, respectively for 4-and-above), yet a bit more diverse for the stronger criterion (63.8 % and 48.9 %, respectively for 5-and-above). The responses on responsibility indicate that CPS PGs can substantially strengthen this aspect, in particular as the numbers for ‘learned nothing’ are very low. The responses for documentation look comparable and indicate a good learning gain. This may be considered surprising because students practice (written) documentation from their first semester on in homework exercises, individual assignments, and the Bachelor’s thesis. A difference to keep in mind is that these documents are usually prepared for a single reader (usually the teacher), while many PG documents need to be effective for multiple readers towards the project goal. Our observations during the project groups confirm that students can in particular learn a lot from documents that are not well written in their first iteration. The figures on self-management are a bit more balanced (68.1 % and 46.8 %, respectively). Still, it is interesting that graduate students in the Master’s program report that they learned a lot on self-management given that the undergraduate studies already require quite good skills for this aspect.

For the more technical aspects, the results of the survey also do not indicate a strongly misguided PG design. The almost evenly distributed learning experiences wrt. hardware development match the PG design where each group develops a few hardware specialists who learn a lot while other participants are required to be able to at least follow the hardware development. Regarding programming, there is a peak of responses who lean towards ‘learned a lot’ but do not strongly agree. We take this as an indicator that the basic programming education in the previous study plan is well sufficient to participate even in challenging CPS PGs and that the special programming languages still provide learning opportunities for many participants. Similar for testing, here the previous education seems to have been even more successful since it is almost impossible to reach the results that the PGs considered here had without a very good test regime.

About two thirds of the respondents (cf. Fig. 4) selected the highest possible value for relevance of this module. This outcome is particularly remarkable, because the respondents of the survey state their impression in hindsight, after having completed their studies. The answers given to the free text questions further support the overall positive impression (cf. Section 5), among them some notable examples such as “the collaboration with the group members and also external partners strongly corresponds to my current working environment in research”, “the process of the project group was very much based on the development of a real product; I now work in software development and especially the application of process models helped”, and “in retrospect, I feel that the project group was by far the module in which I learned the most.” Responses on the question of fair marking indicate that these PGs can be conducted without sacrificing the need for fair individual marks at the end of the studies.

Differentiating the data on learning experience (cf. Fig. 2) between the three contexts research, development, and teaching shows that the response distributions are not equal. Fig. 5 gives the distributions

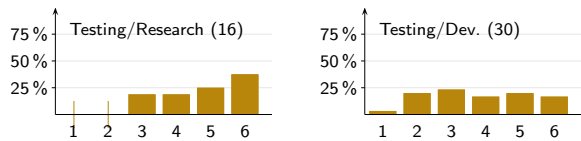


Fig. 5: Learning experience ‘Testing’.

for the aspect ‘Testing’ as an example. It indicates that testing in development contexts has more aspects than can be covered in one PG while the majority is still on the positive side. Still, all responses for context ‘research’ report good learning success which indicates that the CPS PG design does not sacrifice the scientific aspect for the pure engineering part.

The majority (ca. 80 %) of responses states that some development tools or technologies were new in the PG. About half of the former students state that they still use at least one tool or technology that they got to know in their PG in their work environment, some even often. Those who do most often name Matlab Simulink, some even name the hardware oriented Xilinx tools. That only about half of the respondents got to know new project management tools does not come unexpected since the PGs employ lean project management approaches.

7 Conclusion

In this paper, we report the results from a survey to evaluate the effectiveness of our design of a project group that emphasises Cyber-Physical Systems education. The survey was conducted among alumni who participated in one of nine previously offered PGs. The survey had a remarkable response rate of over 72 %, given that the PGs ran many years back for some respondents. The responses collected here are particularly valuable since they come from former students who can assess the learning experience from the perspective of their current working environment. The results confirm that PGs on CPS provide good learning experiences with the main learning objectives focusing on aspects of teamwork and working with more complex problems. A large majority considers the course to be relevant or very relevant in the overall Master’s curriculum.

Acknowledgments. The authors like to thank all students who participated in our PGs. Furthermore, we would like to thank all former students that participated in our survey.

References

- [Be22] Betz, J.; Zheng, H.; Zang, Z.; Sauerbeck, F.; Walas, K.; Dimitrov, V.; Behl, M.; Zheng, R.; Biswas, J.; Krovi, V.; Mangharam, R.: Teaching Autonomous Systems Hands-On: Leveraging Modular Small-Scale Hardware in the Robotics Classroom./, Sept. 2022.
- [BS12] Bauer, K.; Schneider, K.: Teaching cyber-physical systems: A programming approach. In: WESE. Pp. 1–8, 2012.
- [Go15] Gober, P.: Experiences with a Project to Design Autonomous Slotcars in a Mechatronics Master’s Program. In: WESE. WESE’15, ACM, 2015, URL: <https://doi.org/10.1145/2832920.2832925>.
- [Li13] Liebehenschel, J.: Software-Engineering Projekte in der Ausbildung an Hochschulen-Konzept, Erfahrungen und Ideen. In: SEUH. Pp. 27–34, 2013.
- [LL07] Ludewig, J.; Lichter, H.: Software Engineering - Grundlagen, Menschen, Prozesse, Techniken. dpunkt.verlag, 2007.
- [LSJ12] Lee, E. A.; Seshia, S. A.; Jensen, J. C.: Teaching embedded systems the Berkeley way. In: WESE. Pp. 1–8, 2012.
- [Ma20] Marwedel, P.; Mitra, T.; Grimheden, M. E.; Andrade, H. A.: Survey on Education for Cyber-Physical Systems. IEEE Design & Test 37/6, pp. 56–70, 2020.
- [Ri17] Ringert, J. O.; Rumpe, B.; Schulze, C.; Wortmann, A.: Teaching agile model-driven engineering for cyber-physical systems. In: ICSE-SEET. IEEE, pp. 127–136, 2017.
- [Sc11] Schmedding, D.: Teamentwicklung in studentischen Projekten. In: SEUH. Pp. 16–20, 2011.
- [Sc15] Schlender, H.; Schreiner, S.; Metzdorf, M.; Grüttner, K.; Nebel, W.: Teaching mixed-criticality: Multi-rotor flight control and payload processing on a single chip. In: WESE. Pp. 1–8, 2015.
- [St20] Stollenwerk, A.: An Embedded Graduate Lab Course with Spirit. In (Chamberlain, R.; Edin Grimheden, M.; Taha, W., eds.): CyPhy/WESE. Springer International Publishing, pp. 247–263, 2020.
- [TH16] Törngren, M.; Herzog, E.: Towards Integration of CPS and Systems Engineering in Education. In: WESE. WESE ’16, ACM, 2016, URL: <https://doi.org/10.1145/3005329.3005335>.
- [Wa15] Wade, J.; Cohen, R.; Blackburn, M.; Hole, E.; Bowen, N.: Systems Engineering of Cyber-Physical Systems Education Program. In: WESE. WESE’15, ACM, 2015, URL: <https://doi.org/10.1145/2832920.2832927>.

Digitale Prüfungen für Softwareentwicklung im „Bring Your Own Device, Open Book, Open Web“-Format

Axel Böttcher, Veronika Thurner¹

Abstract: Im Sinne des praxisnahen, kompetenzorientierten Prüfens bieten sich zur Erhebung des studentischen Leistungsstandes in der Softwareentwicklung praktische Prüfungen an, mit Nutzung der für die Studierenden individuell gewohnten integrierten Entwicklungsumgebung (IDE) und freier Recherchemöglichkeit. Das Format der „Bring Your Own Device, Open Book, Open Web“-Prüfung wird diesem Anspruch gerecht. Dabei arbeiten die Studierenden auf ihren eigenen Geräten gegen vorgegebene git-Repositories, bei vollem Zugriff auf Wissensressourcen und Internet. Wir haben dieses Prüfungsformat sowohl im Rahmen von Präsenz- als auch von Fernprüfungen erprobt und auf Stärken und Schwächen analysiert.

Als eine der wesentlichen Herausforderungen erweist sich das Finden geeigneter Aufgaben, die einen angemessenen Schwierigkeitsgrad aufweisen und deren Lösung sich dabei nicht ohne weiteres recherchieren lässt. Wir beschreiben zentrale Randbedingungen des Prüfungsformates, insbesondere mit Blick auf die technische und organisatorische Durchführbarkeit. Ergänzend diskutieren wir verschiedene Aufgabenformate unter Gesichtspunkten wie adressierte Kompetenzebene, Korrigierbarkeit und Sicherheit gegen Täuschungsversuche.

Keywords: Digitale Prüfungen, Fernprüfungen, Programmierausbildung

1 Motivation und Ausgangsbasis

Im Rahmen der Programmierausbildung an der Fakultät für Informatik und Mathematik der Hochschule München nutzen die Studierenden in den Praktika zur Softwareentwicklung eine moderne, integrierte Entwicklungsumgebung (IDE), wie sie auch in der Praxis der industriellen Softwareentwicklung zum Einsatz kommen. Des Weiteren lernen die Studierenden die für die Praxis ebenfalls hoch relevante Fähigkeit, zur Lösung von Problemen auf einschlägigen Plattformen, Foren bzw. Q&A-Seiten der Community (z. B. API-Beschreibungen², Stack Exchange³, ...) oder auch über klassische Suchmaschinen zu recherchieren, sich mit den dort gefundenen Tipps und Informationen auseinanderzusetzen und daraus die benötigten Erkenntnisse zu ziehen.

Eine Erhebung des Leistungsstands als schriftliche Prüfung, bei der per Stift Quelltext auf Papier geschrieben wird, ist nicht geeignet, um den Erreichungsgrad der angestrebten

¹ Hochschule München, Fakultät für Informatik und Mathematik, Lothstraße 64, D-80335 München, vorname.nachname@hm.edu

² beispielsweise <https://docs.oracle.com/en/java/javase/19/docs/api/index.html>

³ speziell stackoverflow <https://stackoverflow.com/>

Kompetenzen zu überprüfen. Problematisch ist dabei zum einen der Medienbruch, zum anderen aus didaktischer Sicht insbesondere auch der durch den Medienbruch bedingte Verstoß gegen das Prinzip des Constructive Alignment [BT11]. Gemäß diesem Prinzip sind Lernziele und Prüfungsziele aufeinander abzustimmen und außerdem Lehr-Lernmethoden so zu wählen, dass sie auf das Erreichen der Lernziele (und damit auch der Prüfungsziele) ausgerichtet sind. Beides wäre verletzt, wenn am Ende eines praxisnahen Praktikums in der Softwareentwicklung ausschließlich eine klassische schriftliche Prüfung auf Papier stehen würde.

Entsprechend sind für die Softwareentwicklung praxisnahe Prüfungsformate zu finden, mit denen wir das prüfen können, was wir auch lehren und die Studierenden üben, nämlich die Entwicklung von Software mittels einer IDE. Die Prüfungsformate müssen dabei sicherstellen, dass sie diverse Rahmenbedingungen zwingend einhalten, wie beispielsweise Rechtssicherheit, Korrigierbarkeit, Täuschungsprävention sowie verschiedene Aspekte der technischen und der organisatorischen Durchführbarkeit.

Um praxisnahe Prüfungen auf der Basis von einzusetzenden Softwarewerkzeugen umzusetzen wurde an der Hochschule München EXaHM [Ba21, S. 73f] entwickelt, ein Automatisierungs-Framework für digitale Prüfungen. EXaHM ermöglicht es, den Studierenden eine definierte digitale Prüfungsumgebung bereitzustellen. Dabei legen die Prüfenden fest, welche Softwarewerkzeuge in welcher Konfiguration während der Prüfung nutzbar sind. Gleichzeitig wird über EXaHM die freie Nutzbarkeit des Internets unterbunden, um Täuschungsversuche zu erschweren. Ursprünglich wurde EXaHM für die Durchführung von praktischen Prüfungen in Präsenz entwickelt. Mit der Corona-Pandemie wurde EXaHM so weiterentwickelt, dass es auch die Durchführung von Fernprüfungen ermöglicht.

Gegenüber klassischen Papierprüfungen verbessert EXaHM das Constructive Alignment von Prüfungen in der Softwareentwicklung bereits erheblich [BTZ; HHB]. Einige Probleme bleiben jedoch weiterhin bestehen. Problematisch ist zum einen die zentrale Vorgabe der nutzbaren Softwarekonfiguration einheitlich für alle Teilnehmenden der Prüfung. Das hat zur Folge, dass Studierende, die im Praktikum die von ihnen verwendete IDE nach ihren Bedürfnissen angepasst haben (wie dies in der professionellen Programmierpraxis gang und gäbe ist), in der Prüfung mit einer Arbeitsumgebung konfrontiert werden, die nicht ihren individuellen ergonomischen Bedürfnissen entspricht und in der Bedienung von dem abweicht, was die Studierenden geübt haben und damit gewohnt sind. Die Möglichkeit, sich vor Beginn der Prüfung in die Prüfungsumgebung einzuwählen und diese auszuprobieren entschärft diese Problematik nur marginal – und ersetzt nicht die über ein Semester hinweg erarbeitete Fingerfertigkeit.

Ebenfalls problematisch ist die Einschränkung der Nutzbarkeit des Internets durch EXaHM. Entwickler:innen nutzen in der Entwicklungspraxis eine Vielzahl von Wissensquellen (siehe oben), die ausschließlich via Internet verfügbar sind. Praxistauglich ausgebildete Studierende sollten in der Lage sein, diese Quellen sinnvoll zu nutzen und werden daher im Rahmen der Programmierausbildung an die systematische Nutzung dieser Quellen hingeführt.

Entsprechend ist es im Sinne des Constructive Alignment durchaus sinnvoll, derartige Quellen als Hilfsmittel für eine praxisnahe Prüfung im Bereich Softwareentwicklung zuzulassen. Für Prüfungen, die neben papierbasierten Hilfsmitteln auch Internet-Quellen zulassen, hat sich der Begriff OBOW bzw. „Open Book, Open Web“ eingebürgert [We21; WW09].

Die Corona-Pandemie hat sowohl beim Lehren als auch beim Prüfen einen Digitalisierungsschub erzwungen und viele aus der Not geborene, kreative und konstruktive Lösungen hervorgebracht. Aufgrund des immensen Zeitdrucks insbesondere zu Beginn der Pandemie hatten viele dieser Lösungen den Charakter von „emergency remote teaching“ [Ho20] bzw. „emergency remote assessments“. Beim Übergang ins postpandemische Lehren und Prüfen ist kritisch zu bewerten, welche der entwickelten Ansätze längerfristig tragfähig ist.

2 Ziele

In dieser Arbeit stellen wir ein digitales „Bring Your Own Device, Open Book, Open Web“-Prüfungsformat (BYOD+OBOW-Prüfung) vor, das wahlweise als Präsenzprüfung oder als Fernprüfung durchführbar ist und bei Bedarf einen schnellen Wechsel zwischen diesen Durchführungsmodi ermöglicht. Auf der Grundlage der bisher mit diesem Format gemachten Erfahrungen teilen wir Tipps und Tricks ebenso wie Erkenntnisse zu den mit diesem Format verbundenen Herausforderungen, insbesondere:

- Technische und organisatorische Durchführbarkeit
- Finden geeigneter Aufgabentypen für die zu adressierenden Kompetenzebenen
- (Automatische) Korrigierbarkeit
- Erschweren von Täuschungsversuchen (Unterschleif)

3 Organisatorische und technische Durchführung

Bei der Durchführung einer BYOD+OBOW-Prüfung sind diverse organisatorisch-rechtliche Rahmenbedingungen zu berücksichtigen, deren Umsetzung technologische Hilfsmittel erfordert. Um zuverlässig einen reibungslosen Ablauf der Prüfung zu gewährleisten und aufseiten der Prüfenden den last-minute-Stress vor der Prüfung zu reduzieren empfiehlt es sich, möglichst viele Aspekte der Prüfungsdurchführung mit deutlichem Vorlauf vor dem eigentlichen Prüfungstermin aufzusetzen.

3.1 Durchführungsmodus

Eine BYOD+OBOW-Prüfung lässt sich prinzipiell wahlweise als Präsenzprüfung oder als Fernprüfung durchführen.

Beim Durchführungsmodus *Präsenzprüfung* werden die Studierenden für die Prüfung auf mehrere Räume der Hochschule verteilt. Dort arbeiten Sie dann entweder an ihren eigenen, mitgebrachten Notebooks oder an einem der hochschuleigenen Laborrechner. Im letzteren Fall nutzen die Studierenden die Standard-Installation der Hochschule, wie sie auch auf den Laborrechnern verfügbar ist, die während des Praktikums genutzt werden konnten.

Beim Durchführungsmodus *Fernprüfung* nehmen die Studierenden von einem Ort ihrer Wahl aus an der Prüfung teil, beispielsweise von zu Hause aus. (Eine Teilnahme aus Räumlichkeiten der Hochschule ist prinzipiell auch möglich und wird insbesondere von Studierenden genutzt, die zu Hause schlechtes Internet und/oder keine ruhige Arbeitsumgebung haben.) Gemäß der geltenden gesetzlichen Regelungen wird per Videokonferenz eine zumindest rudimentäre Prüfungsaufsicht geführt [HR23].

In der Fernprüfung nutzen die Studierenden in der Regel ihren eigenen Rechner. Bei Bedarf stellt die Fakultät Leihgeräte bereit, mit denen die Studierenden über das Semester arbeiten und dann auch die Prüfung ablegen können.

In beiden Durchführungsmodi ist von den Studierenden eine Erklärung abzugeben, dass sie die Prüfung selbständig bearbeiten. Bei der Präsenzprüfung wird diese Eigenständigkeitserklärung in Papierform verteilt, beispielsweise auf der Rückseite der Angabe, von den Prüflingen manuell unterzeichnet und dann am Ende der Prüfung von den Aufsichten eingesammelt.

Beim Durchführungsmodus Fernprüfung lässt sich die Eigenständigkeitserklärung dadurch abbilden, dass der eigentlichen Prüfung ein Moodle-Test vorgeschaltet wird, in dem die Studierenden ihre Eigenständigkeitserklärung durch ankreuzen bestätigen.

Des Weiteren schreibt die Bayerische Fernprüfungserprobungsverordnung⁴ vor, dass „für die Studierenden die Möglichkeit bestehen [soll], die Prüfungssituation in Bezug auf die Technik, die Ausstattung und die räumliche Umgebung im Vorfeld der Prüfung zu erproben“ (BayFEV §3, Satz 3). Die Prüfenden sind also verpflichtet, diese technischen Probeläufe rechtzeitig im Vorfeld der Prüfung einzuplanen.

3.2 Arbeitsbereiche zur Verwaltung der studentischen Artefakte

Bei einer digital durchgeführten Prüfung müssen die von den Studierenden erzeugten Artefakte so abgelegt werden, dass sie den einzelnen Prüflingen zweifelsfrei zugeordnet werden können. Nach Abschluss der Prüfung müssen diese Artefakte weiterhin zuverlässig verfügbar sein. Außerdem darf es für die Studierenden nicht möglich sein, die von ihnen erzeugten Artefakte nach Ablauf der Arbeitszeit zu ändern. Dafür ist es nützlich, die studentischen Artefakte mit Zeitstempeln zu versehen.

⁴ BayFEV vom 16. September 2020 (GVBl. S. 570, BayRS 2210-1-1-15-WK)

Um diese Anforderungen umzusetzen wird jedem bzw. jeder Studierenden als Arbeitsbereich für die Prüfung ein eigenes git-Repository zur Verfügung gestellt. Organisatorisch können diese Repositories deutlich vor der Prüfung angelegt werden. Als technische Basis kommt prinzipiell jede zentral gehostete git-Server-Instanz in Frage. Wir sind an unserer Hochschule in der komfortablen Situation, dass alle unsere Studierenden automatisch einen Account auf der gitlab-Instanz des Leibniz-Rechenzentrums der Bayerischen Akademie der Wissenschaften besitzen, der sich dafür datenschutzkonform nutzen lässt. Denkbar ist selbstverständlich grundsätzlich auch die Verwendung anderer Dienste wie `github.com`, `bitbucket.com` oder `gitlab.com` etc.

Deutlich vor dem eigentlichen Prüfungstermin legen die Prüfenden in den studentischen Repositories die Verzeichnisstruktur für das Programmierprojekt der Prüfung an. Konkret geben wir für ein Java-Projekt bereits den `src`-Ordner vor sowie ein Package mit einer Testklasse und einem simplen JUnit-Test `assertTrue(1 == 1)`.

Nachdem die Repositories entsprechend eingerichtet sind, erhalten die Studierenden die erforderlichen Zugriffsrechte für ihr jeweiliges Repository und bekommen anschließend automatisch eine Einladung per E-Mail, etwa zwei Wochen vor dem eigentlichen Prüfungstermin. Daraufhin können sich die Studierenden in Ruhe vor der Prüfung in ihrer eigenen Entwicklungsumgebung das Projekt für die Prüfung einrichten und beispielsweise die erforderlichen Test-Bibliotheken einbinden. Durch diesen Vorlauf lassen sich Probleme mit dem Erstellen des Projekts und dem Einbinden von Test-Bibliotheken während der eigentlichen Prüfungszeit weitest gehend vermieden.

3.3 Ausgeben der Aufgabenstellung

Zu Beginn der Prüfung müssen die Aufgabenstellung sowie von den Prüfenden vorgegebener Quelltext den Studierenden verfügbar gemacht werden. Dieser Prozess ist zeitkritisch, da aus rechtlichen Gründen entweder die Prüfung für alle Teilnehmenden zeitgleich starten muss oder zumindest eine gleiche Bearbeitungszeit für alle Prüflinge garantiert sein muss.

Bisher haben wir Erfahrungen mit zwei Alternativen gesammelt:

1. Die Vorgaben werden in einem verschlüsselten zip-Archiv bereits deutlich vor Prüfungsbeginn in die Repositories gepusht. Die Studierenden erhalten dann zu Beginn der Bearbeitungszeit das Passwort, das zum Entschlüsseln des zip-Archivs erforderlich ist.
2. Die Vorgaben werden zu Beginn der Bearbeitungszeit per Shell-Skript auf alle Repositories gepusht.

Probelaufe haben ergeben, dass viele Studierende bei der Variante mit dem zip-Archiv massive Schwierigkeiten damit hatten, die Archive so zu entpacken, dass die entpackten Dateien im Projekt an der richtigen Stelle lagen und somit von der IDE korrekt verarbeitet werden konnten. Dieses Vorgehen wurde zwar vor der Prüfung im Praktikum gezeigt

und von den Studierenden selbst durchgeführt, konnte aber im Rahmen des technischen Probelaufs nicht von allen Prüflingen abgerufen werden.

Die Variante mit dem Push via Shell-Skript hat dagegen aufseiten der Studierenden reibungslos funktioniert, ist aber natürlich anfälliger für akute Netzwerkprobleme.

Bei der Variante Präsenzprüfung ist es empfehlenswert, die Aufgabenstellung auf Papier auszuteilen, mit der von den Studierenden zu unterschreibenden Erklärung zur selbstständigen Bearbeitung auf der Rückseite des Angabenblattes. Das ermöglicht zum einen manuelle Unterschriften der Studierenden und stellt zum anderen sicher, dass das Angabenblatt am Ende der Prüfung zuverlässig abgegeben bzw. wieder eingesammelt wird. Damit lässt sich vermeiden – oder zumindest erschweren – dass die konkrete Aufgabenstellung während der Prüfung an Dritte zum Zwecke der Inanspruchnahme von Fremdhilfe weitergegeben wird. Ebenso lässt sich die Weitergabe der Aufgabenstellung nach der Prüfung innerhalb der Studierendenschaft erschweren.

3.4 Beginn der Prüfung

Aus rechtlichen Gründen muss die Prüfung entweder für alle Prüflinge synchron starten und enden, oder sichergestellt sein, dass allen Prüflingen exakt die gleiche Bearbeitungszeit zur Verfügung steht. Falls physische Prüfungsräume oder virtuelle Breakout-Räume der Prüfungsumgebung nicht exakt synchronisiert werden können, muss daher zumindest für jede Teilkohorte der genaue Start- und Endzeitpunkt der Prüfung protokolliert werden.

Werden verschlüsselte Archive verwendet, um die Vorgaben zur Prüfung zu verteilen, so ist der Startzeitpunkt der Prüfung definiert durch die Bekanntgabe des Kennworts, das zum Entschlüsseln der Archive notwendig ist.

Werden dagegen die Vorgaben zur Prüfung zum Prüfungsbeginn unverschlüsselt gepusht, so landen die Daten nicht exakt gleichzeitig in den Repositories der Studierenden. Bei dieser Variante muss also durch die Aufsichten ein synchroner Start der Bearbeitungszeit für alle Prüflinge sichergestellt werden. Weitere Unwägbarkeiten dieser Variante sind eventuell auftretende Netzwerkprobleme. Denkbar wäre außerdem, dass auf einzelnen Repositories der Push der Prüfenden fehl schlägt, weil die zugehörigen Studierenden unerlaubter Weise dort selbst schon einen Push durchgeführt haben. In unseren Durchführungen ist bisher noch keine dieser beiden Problemsituationen aufgetreten.

3.5 Abgeben studentischer Artefakte während der Prüfung

Die Studierenden legen die von ihnen während der Prüfung erzeugten Artefakte selbst aktiv in ihrem jeweiligen Repository ab. Artefakte werden also *nicht* automatisch gesichert. Die Nutzung von git-Repositories wurde während des Praktikums über das Semester hinweg

kontinuierlich geübt. Studierende, die sich aktiv am Praktikum beteiligt haben, sollten die dafür notwendigen Schritte also beherrschen.

3.6 Ende der Prüfung

Ist der Beginn des Bearbeitungszeit klar definiert, so kann auch das Prüfungsende als ein eindeutig definierter Zeitpunkt festgestellt werden. Push-Operationen nach diesem Zeitpunkt werden nicht mehr gewertet. Alternativ ist es auch möglich, am Ende der Prüfung den Studierenden auf ihren Repositories die Schreibrechte zu entziehen. Bei dieser Variante ist es aus rechtlichen Gründen zwingend erforderlich, Durchführung und Zeitpunkt des Entziehens der Schreibrechte rechtssicher zu dokumentieren, beispielsweise durch log-Files des Servers.

Zu beachten ist ferner, dass die git-Server nicht den Zeitpunkt des Push protokollieren, sondern lediglich die Commit-Vorgänge mit Zeitstempel versehen. Durch Manipulation der Uhr ihres lokalen Systems könnten Studierende sich somit einen Vorteil verschaffen und nach dem offiziellen Ende der Prüfung noch Lösungen pushen. Es empfiehlt sich daher, den Prüflingen unmittelbar nach Ende der Bearbeitungszeit die Schreibrechte auf die Repositories zu entziehen. Dies ist nicht immer durch skriptbare Kommandozeilenbefehle möglich, sodass hier gegebenenfalls noch weitere Entwicklungsarbeit nötig ist.

Zu beachten ist, dass einige Studierende im Rahmen von Nachteilsausgleichen eine individuell verlängerte Bearbeitungszeit zuerkannt bekommen. Diese ist entsprechend bei der Durchführung zu berücksichtigen.

4 Definieren geeigneter Aufgabenstellungen

Im Sinne des Constructive Alingment müssen die Prüfungsaufgaben in Einklang stehen zu den Lernzielen, die mit der Lehr-Lernveranstaltung verfolgt werden. Lernziele für Softwareentwicklung formulieren wir gemäß den Kompetenzebenen der Taxonomie von Bloom in der Überarbeitung von Anderson und Kratwohl [An01], wie im Detail dargestellt in [Th15]. Im Rahmen der Lehr-Lernveranstaltung kommunizieren wir diese Lernziele aktiv an die Studierenden. Im Sinne des Constructive Alingment sind Lehr-Lernmethoden und insbesondere die Praktikumsaufgaben während des Semesters auf diese Lernziele abgestimmt. Gleiches müssen entsprechend auch die Prüfungsaufgaben leisten.

4.1 Niedrige Kompetenzebenen

Eine Analyse von Fehlermustern liefert Indizien dafür, dass das Beherrschen der Fachsprache eine wichtige Voraussetzung für gute Leistungen in Softwareentwicklung ist [Bö16]. Dies umfasst zum einen das korrekte eigene Verwenden von Fachbegriffen, zum anderen die kor-

rekte Interpretation dieser Fachbegriffe, wenn diese beispielsweise in einer Aufgabenstellung geschrieben stehen.

Im Kontext der Taxonomie von Anderson und Krathwohl entsprechen diese Fähigkeiten den Kompetenzebenen 1 (Erinnern) und 2 (Verstehen). Zugehörige Lernziele sind beispielsweise:

- *Kompetenzebene 1: Erinnern*
Die Studierenden schreiben die konkrete Syntax eines programmiersprachlichen Konstruktes korrekt auf und halten dabei die Syntaxkonventionen ein.
- *Kompetenzebene 2: Verstehen*
Die Studierenden erklären in eigenen Worten die Bedeutung eines programmiersprachlichen Konstruktes.

Insbesondere bei OBOW-Prüfungen macht es nur bedingt Sinn, diese Fähigkeiten direkt explizit abzu prüfen, da die Antworten leicht recherchiert werden können. Zielführender und insbesondere auch realitätsnäher ist es, das Beherrschen der Fachsprache im Kontext von praktischen Aufgabenstellungen auf integrierte Weise zu überprüfen. Ein typisches Beispiel für eine entsprechende Aufgabenstellung ist:

- *Aufgabe zu Fachsprache, Kompetenzebenen 1 und 2*
Schreiben Sie eine Klassenmethode, die ... *<Beschreibung einer sehr einfachen Funktionalität>*. Halten Sie bei der Implementierung die Syntaxkonventionen ein.

Die Einhaltung von Syntaxkonventionen fließt dabei in die Bewertung mit ein, da sie explizit als Lernziel der Veranstaltung formuliert ist und im Praktikum systematisch eingeübt wurde.

4.2 Mittlere Kompetenzebenen

Unit-Testing ist in der professionellen Softwareentwicklung essenziell wichtig und wird daher auch explizit von Lernzielen adressiert. Beispiele sind:

- *Kompetenzebene 3: Anwenden*
Die Studierenden erstellen schematisch grundlegende Testfälle.
- *Kompetenzebene 4: Analysieren*
Die Studierenden untersuchen, welche Rand- und Normalfälle in einer gegebenen Aufgabenstellung auftreten, fassen diese zu geeigneten Äquivalenzklassen zusammen und definieren dazu passende Testfälle als Repräsentanten.

Eine entsprechende Aufgabenstellung gibt produktiven Quelltext vor, den die Studierenden dann durch geeignete Unit-Tests absichern sollen.

4.3 Hohe Kompetenzebenen

Lernziele auf den hohen Kompetenzebenen sind beispielsweise:

- *Kompetenzebene 5: Evaluieren*
Die Studierenden wägen systematisch ab, welche Kontrollstruktur am besten geeignet ist, um eine bestimmte Anforderung umzusetzen.
- *Kompetenzebene 6: Kreieren*
Die Studierenden entwickeln für ein einfaches Problem aus einer gegebenen Anforderungsspezifikation heraus einen Algorithmus und implementieren diesen unter Verwendung geeigneter Kontrollstrukturen.

Eine korrespondierende Aufgabenstellung beschreibt eine Funktionalität, die von den Studierenden umzusetzen ist:

- *Aufgabe zur Implementierung und Algorithmen, Kompetenzebenen 5 und 6*
Schreiben Sie im Package a2 eine Klasse mit einer geeigneten Klassenmethode mit einem Parameter vom Typ String und Returntyp boolean. Die Methode soll prüfen, ob die in der übergebenen Zeichenkette enthaltenen runden Klammern geordnete Paare bilden.

Argument an die Methode	Returnwert
"	true
"Blau_Laber"	true
"(FooBar)(bla)"	true
"(Foo(Bar)((bla())())((())))"	true
"()()()()())"	true
"((()())"	false
")(""	false
"((((""	false
"((()"	false
"(Foo(Bar)(bla())())"	false
"(Foo())(Bar)((Foobar)"	false

Bei einer „Open Web“-Prüfung müssen derartige Aufgabenstellungen so formuliert sein, dass eine Internet-Recherche nach passenden Lösungen ergebnislos bleibt oder zumindest hohen Aufwand erfordert. Die Schwierigkeit besteht im Wesentlichen darin, die Beschreibung so zu halten, dass eine direkte Eingabe von Teilen der Aufgabenstellung bzw. sogar von einzelnen Schlagworten eine Suchmaschine nicht direkt zu einer Lösung führen. Verboten sind also Worte wie „Sortieren“ oder „Palindrom“, oder konkret für unser Beispiel der Begriff „Klammernchecker“. Für die Prüfenden ist die Situation wie im Spiel „Tabu“, wo Begriffe erklärt werden müssen, ohne den jeweiligen Begriff oder Bestandteile davon zu nennen [He90].

5 Auswertung der Prüfungsergebnisse

Am Ende der Prüfung liegen die studentischen Artefakte in elektronischer Form vor. Für eine effiziente Korrektur ist es hilfreich, zunächst die studentischen Lösungen aus den individuellen studentischen Repositories in ein einziges Projekt des bzw. der Prüfenden zusammenzutragen und diese dort teilautomatisiert auszuwerten. Damit die studentischen Artefakte auch nach diesem Kopiervorgang eindeutig den erhebenden Studierenden zugeordnet werden können, kopieren wir dafür die Lösung jedes bzw. jeder Studierenden in ein Package im Projekt des bzw. der Prüfenden. Der Bezeichner dieses Packages leitet sich aus dem Namen des bzw. der Studierenden ab. (Die dafür benötigte Information zu den Namen der Studierenden lässt sich automatisch ermitteln aus den Bezeichnern der studentischen Repositories.)

5.1 (Automatische) Korrektur und Bewertung

Dadurch, dass die studentischen Artefakte am Ende der Prüfung in elektronischer Form vorliegen, ergibt sich die Möglichkeit, die Auswertung dieser Artefakte zumindest teilweise automatisiert zu unterstützen. Die genauen Mechanismen hängen dabei ab von der konkreten Art der Aufgabenstellung.

Aufgaben, in denen Studierende gegen vorgegebenen Quelltext selbst Unit-Tests schreiben (siehe Abschnitt 4.2), lassen sich durch automatisierte Coverage-Messungen bewerten, entsprechend der in der Aufgabe gestellten Anforderung C0, C1 oder Mutation Coverage.

Eine von den Prüflingen zu implementierende Klassenmethode (siehe Abschnitt 4.3) lässt sich bewerten über Testfälle, die mit überschaubarem Aufwand per Skript-Programmierung automatisiert an die von den Studierenden gewählten Bezeichner angepasst und dann ausgeführt werden können. Wenn es zu jeder Äquivalenzklasse möglicher Eingabewerte einen Test gibt, lassen sich aus der Anzahl fehlschlagender Tests automatisch Bewertungspunkte ableiten. In unserem Beispiel mit fünf Äquivalenzklassen wurden ab drei fehlschlagenden Tests 0 Punkte vergeben, weil bei einer Methode mit Rückgabety `boolean` grüne Testfälle mit einem simplen `return true`; zu erreichen sind.

Das Erkennen, ob eine Methode wie vorgegeben als Klassen- und nicht als Objektmethode implementiert wurde (siehe Aufgabe in Abschnitt 4.1), lässt sich ebenfalls gut mit Skripten automatisieren. An anderen Stellen waren dagegen manuelle Korrekturschritte erforderlich.

Studierende, deren git-Repository am Ende der Prüfungszeit leer ist, haben die Prüfung automatisch nicht bestanden. Dabei lässt sich nicht überprüfen, ob die Studierenden auch in ihrer lokalen Arbeitsumgebung keine Artefakte erzeugt hatten oder ob sie am einchecken dieser Artefakte ins git-Repository gescheitert sind.

5.2 Erschweren von Täuschungsversuchen

Generell sind im deutschen Hochschulraum Täuschungsversuche bei Prüfungen ein Thema, unabhängig von der Durchführungsform. In der Studie von [Ja21] mit 1608 Studierenden verschiedener Hochschultypen in Deutschland geben 31,7% der Befragten zu, bei Präsenzprüfungen Täuschung versucht bzw. durchgeführt zu haben, im Vergleich zu 61,4% bei Fernprüfungen. Der Begriff der Täuschung umfasst dabei sowohl die Nutzung von nicht erlaubten Hilfsmitteln als auch die unerlaubte Inanspruchnahme von Hilfe durch andere Personen (Fremdhilfe).

Da bei „Open Book, Open Web“-Prüfungen prinzipiell alle Ressourcen (außer Fremdhilfe durch Personen) genutzt werden dürfen ist diese Thematik im hier beschriebenen Prüfungsformat unkritisch, da explizit mit eingeplant. Durch die freie Recherche-Möglichkeit ergeben sich jedoch besondere Anforderungen an die Aufgabenstellung (siehe Abschnitt 4.3). Täuschung durch Fremdhilfe muss dagegen beachtet werden.

Bei unseren bisherigen Durchführungen von BYOD+OBOW-Prüfungen haben wir keine Kommunikationsversuche der Studierenden untereinander festgestellt. Das bedeutet, es gab weder Verdachtsmomente im Rahmen der (virtuellen) Prüfungsaufsicht noch eindeutig identifizierbare Plagiate bei den abgegebenen Artefakten. Zur automatischen Plagiatserkennung in den studentischen Artefakten nutzen wir MOSS (für „Measure Of Software Similarity“)⁵, ein System zur Erkennung von Ähnlichkeiten in Software.

Grundsätzlich besteht auch die Möglichkeit, dass die Studierenden mit Dritten kommunizieren, also mit nicht selbst an der Prüfung beteiligten Personen. Insbesondere bei einer Durchführung als Fernprüfung kann nicht verhindert werden, dass Studierende per Chat oder IDE-Plugin (wie z. B. codetogether) mit Anderen kommunizieren.

Bei der Durchführungsvariante als Präsenzprüfung lässt sich Täuschung durch Fremdhilfe dadurch erschweren, dass die Aufgabenstellung auf Papier ausgeteilt wird. Im Vergleich zu elektronisch verfügbaren Aufgabenblättern erschwert eine Aufgabenstellung auf Papier eine Täuschung durch Fremdhilfe, weil in der Papier-Konstellation die Aufgabenstellung ggf. erst einmal abgetippt bzw. abfotografiert werden muss, bevor sie via Internet mit den potenziellen Helfenden geteilt werden kann. Es steht zu hoffen, dass die Präsenzaufsicht ein Abfotografieren bemerken und unterbinden würde. Abtippen und Vermailen ist weniger auffällig, kostet dafür aber entsprechend mehr Zeit.

6 Zusammenfassung und Ausblick

Nach den Erfahrungen der letzten Semester hat sich das Format der BYOD+OBOW-Prüfungen bewährt. Bei entsprechender organisatorischer und technischer Vorbereitung läuft die Durchführung reibungslos, egal ob in Präsenz oder als Fernprüfung. Des Weiteren haben die bisherigen Durchführungen nach unserem Eindruck angemessen zutreffend den studentischen Leistungsstand erhoben, der über das Semester hinweg auch in den Praktika zu beobachten war.

Ausbaubar bleibt unser Fundus an Aufgabentypen, die sich für praxisnahe Übungen eignen und außerdem gut per Blickdiagnose oder automatisiert unterstützt korrigierbar sind. Bei Aufgaben, in denen die Prüflinge selbst Unit-Tests entwickeln, wären z. B. als abzugebendes Artefakt Screenshots denkbar, welche die Messergebnisse der Testabdeckung dokumentieren.

Literatur

- [An01] Anderson, L. W.; Krathwohl, D. R.; Airasian, P. W.; Cruikshank, K. A.; Mayer, R. E.; Pintrich, P. R.; Raths, J.; Wittrock, M. C.: A Taxonomy for Learning,

⁵ <https://theory.stanford.edu/aiken/moss/>

- Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. Longman, New York, 2001.
- [Ba21] Bandtel, M.; Baume, M.; Brinkmann, E.; Bedenlier, S.; Budde, J.; Eugster, B.; Ghoneim, A.; Halbherr, T.; Persike, M.; Rampelt, F.; Reinmann, G.; Sari, Z.; Schulz, A.: Digitale Prüfungen in der Hochschule. Whitepaper Version 1.1, https://hochschulforumdigitalisierung.de/sites/default/files/dateien/HFD_Whitepaper_Digitale_Pruefungen_Hochschule.pdf zugegriffen 26.10.2022, 2021.
- [Bö16] Böttcher, A.; Thurner, V.; Schlierkamp, K.; Zehetmeier, D.: Debugging students' debugging process. In: IEEE Frontiers in Education Conference (FIE). S. 1–7, 2016.
- [BT11] Biggs, J.; Tang, C.: Teaching For Quality Learning At University. McGraw-Hill Education, 2011, ISBN: 9780335242757.
- [BTZ] Böttcher, A.; Thurner, V.; Zehetmeier, D.: Alignment of Teaching and Electronic Exams and Empirical Classification of Errors for an Introductory Programming Class. In: IEEE CSEE T, 2020. S. 1–10.
- [He90] Hersch, B.: Tabu, [Brettspiel], Hasbro, Pawtucket, Rhode Island, 1990.
- [HHB] Hammer, S.; Hobelsberger, M.; Braun, G.: Using laboratory examination to assess computer programming competences: Questionnaire-based evaluation of the impact on students. In: IEEE EDUCON 2018. S. 355–363.
- [Ho20] Hodges, C.; Moore, S.; Lockee, B.; Trust, T.; Bond, A.: The Difference Between Emergency Remote Teaching and Online Learning, <https://er.educause.edu/articles/2020/3/the-difference-between-emergency-remote-teaching-and-online-learning> zugegriffen 29.07.2022, März 2020.
- [HR23] Heckmann, D.; Rachut, S.: E-Klausur und Elektronische Fernprüfung, Rechtsfragen der Umstellung von Hochschulprüfungen auf zeitgemäße, digitale Prüfungsformate. Duncker & Humblot GmbH, Berlin, 2023.
- [Ja21] Janke, S.; Rudert, S.; Petersen, Ä. L.; Fritz, T.; Daumiller, M.: Cheating in the Wake of COVID-19: How Dangerous is Ad-hoc Online Testing for Academic Integrity? Computers and Education Open 2/, S. 100055, Okt. 2021.
- [Th15] Thurner, V.; Böttcher, A.; Schlierkamp, K.; Zehetmeier, D.: Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen. In: Tagungsband des 14. Workshops Software Engineering im Unterricht der Hochschulen 2015, Dresden, Deutschland, 26. - 27. Februar 2015. S. 9–20, 2015.
- [We21] Weber-Wulff, D. In: 9. Fachtagung Hochschuldidaktik Informatik (HDI 2021), 15.-17. September 2021 in Dortmund. Dortmund. S. 175–179, 2021.
- [WW09] Williams, J.; Wong, A.: The efficacy of final examinations: A comparative study of closed-book, invigilated exams and open-book, open-web exams. British Journal of Educational Technology 40/, S. 227–236, Feb. 2009.

Nachhaltigkeit als Qualitätskriterium von Software - Den Blick auf ressourcen-sparsame Softwareentwicklung schärfen

Oliver Radfelder,¹ Karin Vosseberg²

Abstract: In der Software Engineering Ausbildung an Hochschulen liegt meist der Schwerpunkt auf Methoden und Verfahren der Konstruktion komplexerer Softwaresysteme. Mit dem vorliegenden Beitrag wird diskutiert, wie Studierende durch analytische Maßnahmen der Qualitätssicherung für eine ressourcen-sparsame Softwareentwicklung sensibilisiert werden können. Den Studierenden werden einfache Werkzeuge an die Hand gegeben, um insbesondere die nicht-funktionalen Eigenschaften eines Softwaresystems zu untersuchen. Bei nicht-funktionalen Eigenschaften spielen neben den neu oder weiterentwickelten Softwarekomponenten auch die Laufzeitumgebung eine wesentliche Rolle, so dass auch diese in die Betrachtung der Qualitätssicherung integriert werden muss und damit auch immer Teil der Entwicklung eines Softwaresystems ist. Anhand eines konkreten Beispiels wird der Einfluss der Laufzeitumgebung auf das Verhalten und dem Ressourcenverbrauch einer Webanwendung mit den Studierenden diskutiert, um den Blick auf das Zusammenspiel zwischen Applikation und Laufzeitumgebung zu schärfen. Zudem eignet sich das Beispiel, um Vorteile von DevOps als Vorgehensmodell erlebbar zu machen.

Keywords: Softwareentwicklung; Softwarekonfiguration; analytische Qualitätssicherung; konstruktive Qualitätssicherung; Messen von Software; nichtfunktionale Qualitätskriterien; Ressourcenverbrauch; Nachhaltigkeit

1 Einleitung

Im Curriculum der Bachelorstudiengänge Informatik und Wirtschaftsinformatik an der Hochschule Bremerhaven wird neben der Programmierausbildung dem Software Engineering mit drei Modulen viel Zeit eingeräumt, um die verschiedenen Aspekte der Entwicklung komplexerer Softwaresysteme aufeinander aufbauend einzuführen: *Software Engineering I* mit Schwerpunkt Modellbildung, *Software Engineering II* mit Schwerpunkt Requirements Engineering und einer prototypischen Umsetzung zur Evaluierung von Requirements sowie *Software Engineering III* mit dem Schwerpunkt Softwarearchitekturen und der Umsetzung einer langlebigen, skalierbaren Webanwendung. Das Thema Qualität von Software spielt in allen drei Veranstaltungen eine große Rolle, so dass in den Veranstaltungen Maßnahmen der Qualitätssicherung auf den unterschiedlichen Ebenen der Softwareentwicklung eingeführt werden, die neben der funktionalen Qualität insbesondere auch die nicht-funktionalen Qualitätsmerkmale von Software betrachten. Ein besonderer Fokus liegt hierbei auf Langlebigkeit, Skalierbarkeit und Ressourcen-Sparsamkeit von Software.

¹ Hochschule Bremerhaven, An der Karlstadt 8, 27568 Bremerhaven oradfelder@hs-bremerhaven.de

² Hochschule Bremerhaven, An der Karlstadt 8, 27568 Bremerhaven kvosseberg@hs-bremerhaven.de

Geschuldet dem gemeinsamen Interesse der Kolleg:innen dieser Veranstaltungen an einer ökologischen und sozialen Nachhaltigkeit [ERV20] in der IT zieht sich das Thema des ressourcen-sparsamen Umgangs in der Softwareentwicklung durch weitere Module des Curriculums. Bereits in der *Studieneingangsphase* werden die Studierenden der Informatik und Wirtschaftsinformatik an eine ressourcen-sparsame Umsetzung von kleinen Automatisierungsketten herangeführt und in Modulen wie *Infrastrukturen* für Informatikstudierende oder *Technik für Wirtschaftsinformatik* lernen sie den Ressourcenverbrauch mit einfachen Werkzeugen zu beobachten, z.B. das Messen der Ausführungszeit einer Funktion, des Durchsatzes paralleler Anfragen oder von Speicherverbrauch. Damit wird auch der Mehrwert aus den Beobachtungen des operativen Betriebs thematisiert. Um das Thema der Nachhaltigkeit explizit in der Kultur des Informatikbereichs an der Hochschule sichtbar zu machen, wurde im Rahmen der Klimawoche an der Hochschule Bremerhaven ein ganztägiger Workshop *Ressourcensparsames Programmieren* angeboten.³

Darauf aufbauend thematisieren wir in dem Wahlpflicht-Modul *Grundlagen der Qualitätsmanagements* auf Basis von funktionalen und nicht-funktionalen Qualitätskriterien [SL19] die verschiedenen Facetten einer nachhaltigen Softwareentwicklung. Neben Energieverbrauch und Ressourcen-Sparsamkeit spielen Themen wie Wartbarkeit genauso wie Langlebigkeit von Software eine wichtige Rolle in der Diskussion um Nachhaltigkeit. In der Veranstaltung wird der Fokus gelegt auf Verfahren der analytische Qualitätssicherung und die Frage, was wir aus diesen Betrachtungen für eine konstruktive Qualitätssicherung gemeinsam lernen können. Im folgenden wird das Ziel der Veranstaltung beschrieben und anhand eines Beispiels thematisiert, wie Studierende an eine analytische Betrachtung von Software und dem Zusammenspiel mit der operativen Laufzeitumgebung der Software mit einem konkreten Versuchsaufbau herangeführt werden können. Hierbei wird ein Aspekt der Nachhaltigkeit - der ressourcen-sparsame Umgang mit Speicher in der genutzten Docker-Laufzeitumgebung - diskutiert. Außerdem fungiert das Beispiel zur Motivation eines DevOps-Vorgehensmodells, weniger im Sinne einer schnelleren Produktauslieferung [KS19] sondern eher in der gewinnbringenden Nutzung des Wissens aus dem operativen Betrieb eines Softwaresystems.

2 Analytische und konstruktive Qualitätssicherung

Im Rahmen des Wahlpflicht-Moduls *Grundlagen des Qualitätsmanagements* haben wir den Schwerpunkt auf das Zusammenspiel von Maßnahmen konstruktiver und analytischer Qualitätssicherung gelegt. Im Wechsel werden analytische Verfahren der Qualitätssicherung vorgestellt und daraufhin diskutiert wie dieses Wissen auch in der Entwicklung von Softwaresystemen genutzt werden kann. Exemplarisch wird mit den Studierenden gemeinsam erarbeitet, wie eine Umsetzung in einer Java-Umgebung aussehen kann.

³ <https://informatik.hs-bremerhaven.de/uerb/Klimawoche/>

In den Diskussionen und der Literatur zu Verfahren der analytischen Qualitätssicherung wird sehr stark auf die funktionalen Eigenschaften von Software fokussiert. Hier sind etablierte Verfahren bekannt, die einen systematischen Zugang zur Qualitätssicherung ermöglichen (siehe z.B. [SL19]). Solche Verfahren wurden in der Veranstaltung auf Basis des Lehrplans zum *Certified Tester Foundation Level* [GTB20] vorgestellt. Es wurde diskutiert, wie die Erkenntnisse aus den analytischen Testverfahren für die Programmierung der Funktionalität von Software genutzt werden können, um typische Fehler zu vermeiden und in Programmbeispielen gemeinsam umgesetzt.

Die Qualitätssicherung nicht-funktionaler Eigenschaften von Softwaresystemen wird in der Praxis oft als *Sonderaufgabe* abgeschoben und geht nicht wirklich in Alltagshandeln der Projekte über. Tests von nicht-funktionalen Eigenschaften, wie beispielsweise Last- und Performance oder effizienter Umgang mit Ressourcen scheinen unabhängig vom Vorgehensmodell kaum eine Rolle zu spielen. Dies spiegeln auch die Ergebnisse der Umfrage *Softwaretest in Praxis und Forschung* aus den Jahren 2011, 2016 und 2020 wider [WVS21]. Im Vergleich über die Jahre ist erkennbar, dass die Bedeutung unterschiedlicher Testarten seit 2011 sogar abnimmt (siehe Abbildung 1).

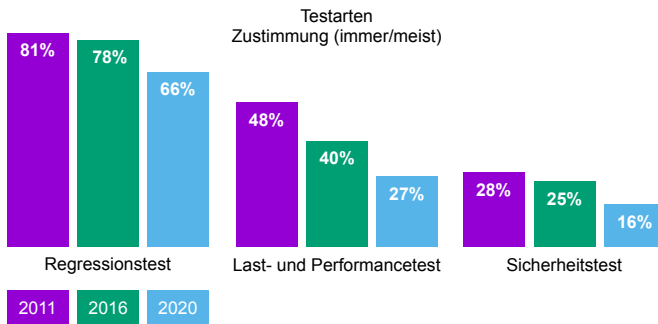


Abb. 1: Bedeutung von unterschiedlichen Testarten in Projekten

Diese Entwicklung erstaunt um so mehr im Kontext agiler Vorgehensmodelle. Mit der Transformation zu agilen Vorgehensmodellen wird die Verantwortung für Qualität in die Projekte gelegt und Maßnahmen der Qualitätssicherung in die Softwareentwicklung integriert. Dies ist auch an einem hohen Grad an Testautomatisierung auf der Unittest-Ebene zu beobachten. Kurze Entwicklungszyklen und die Diskussionen zu DevOps erhöhen die Verbindung zum operativen Betrieb. Es scheint jedoch keine Auswirkungen auf die Bedeutung nicht-funktionaler Qualitätssicherung zu haben.

In Diskussionen mit Projektmitarbeiter:innen aus Unternehmen wird die Lösung von Last- und Performanceproblemen in der horizontalen Skalierung oft mit Parallelisierung und Hinzufügen von weiteren Ressourcen gesehen, ohne den Blick auf die entstehende zusätzliche Komplexität oder einen nachhaltigen Umgang mit Ressourcen zu nehmen. Für eine Untersuchung, ob zusätzliche Ressourcen die beste Lösung zur Erhöhung der Skalierbarkeit ist, ist eine analytische Betrachtung des Gesamtsystems - Applikation und

Laufzeitumgebung - notwendig. Dies scheint jedoch in den Projekten, die den Blick haben auf der Entwicklung neuer Features ihrer (Teil-)Applikation, kaum Raum zu finden. Leider liegen solche Fragestellungen oft auch nicht im Bereich von zentralen Qualitätssicherungsteams, deren Aufgabe ist, eine Laufzeitumgebung zur Verfügung zu stellen und weiterzuentwickeln.

Die Umsetzung nicht-funktionaler Eigenschaften eines Softwaresystems ist jedoch eine wesentliche Aufgabe in der Softwareentwicklung und gehört zu den Aspekten der konstruktiven Maßnahmen der Qualitätssicherung. Werden nicht-funktionale Qualitätseigenschaften eines Systems nicht bereits in der Entwicklung und Umsetzung von Softwarearchitekturen berücksichtigt, wird es schwierig diese Eigenschaften zu erreichen. Dabei spielen gerade die Erkenntnisse aus analytischen Maßnahmen der Qualitätssicherung nicht-funktionaler Eigenschaften des Gesamtsystems und insbesondere dem Zusammenspiel zwischen Applikation und Laufzeitumgebung eine wesentliche Rolle. Sie geben wertvolle Hinweise für die Konstruktion der Softwaresysteme und sollten in das Alltagshandeln von Entwickler:innen verankert werden. Ziel der Veranstaltung ist gerade diese Aspekte der Softwareentwicklung zu adressieren, um anhand von Beispielen gemeinsam mit den Studierenden Lösungsansätze für die Analyseaufgaben zu entwickeln und ein Monitoring bereits während der Softwareentwicklung aufzubauen. Aus diesem Grund haben wir in der Veranstaltung neben den Verfahren der analytischen Qualitätssicherung funktionaler Qualitätseigenschaften auch die Analyse und das Monitoring von Systemen aufgenommen, um nicht-funktionale Eigenschaften während des Entwicklungsprozesses zu untersuchen.

Das im Folgenden beschriebene Problem des *Off-Heap Memory Leaks in Servlet-Containern* ist ein solches konkretes Beispiel, um ein Gesamtsystem zu analysieren und die Ursachen der Probleme zu lokalisieren, die oft nicht unbedingt in der Umsetzung einer Applikation liegen, sondern im Zusammenspiel mit der Laufzeitumgebung. Um dieses Problem zu lösen wird nicht selten die Strategie verwendet eine containerisierte Laufzeitumgebung mit zusätzlichen Ressourcen zu versehen, in diesem Fall zusätzlichem Speicher, und die Umgebung regelmäßig neu zu starten. Unser Ansinnen ist, das dahinterstehende Problem zu verstehen, zu lokalisieren und mit weiteren Lösungsmöglichkeiten zu vergleichen. Gemeinsam mit den Studierenden wurde das beobachtete Phänomen diskutiert und eine Hypothese formuliert. Schritt für Schritt wurde ein Versuchsaufbau entwickelt und umgesetzt, um die Hypothese zu verifizieren. Ziel ist den Studierenden Handlungsoptionen sichtbar zu machen.

Gerade im Kontext der Betrachtung einer ökologischen Nachhaltigkeit in der Entwicklung von Softwaresystemen [GGP22] wird die Analyse des Gesamtsystems ein wichtiger Bestandteil in der Beurteilung dieses Qualitätskriteriums sein, um die Hinweise für eine Einordnung zu nutzen aber auch um in der Weiterentwicklung des Gesamtsystems die Ergebnisse einfließen zu lassen. Ein Aufsetzen eines entsprechenden Monitorings bereits während der Softwareentwicklung unterstützt, dass Veränderungen, die durch Weiterentwicklungen oder Migrationen in eine neue Laufzeitumgebung entstehen, sofort erkannt werden können. Damit begleiten solche Maßnahmen intensiv den Softwareentwicklungsprozess und unterstützen die kontinuierliche Qualitätsverbesserung. Insbesondere im Kontext von *DevOps* entsteht so

eine enge Verbindung zwischen der (Weiter-)Entwicklung von Software und dem Betrieb der Softwaresysteme.

3 Beispiel: Off-Heap Memory Leaks in Servlet-Containern

In der Infrastruktur der Informatik an der Hochschule Bremerhaven werden den Studierenden Docker-Container zur Verfügung gestellt, in denen als Servlet-Engines *tomcat*, *wildfly* und *jetty* installiert sind. Die Java-Versionen 8, 11 und 17 stehen ebenfalls bereit. Die Container selbst werden mit je vier virtuellen CPUs und zwei GB virtuellem Speicher gestartet. Die Container sind sehr bewusst mit beschränkten Ressourcen ausgestattet, um einen ressourcen-sparsamen Umgang in der Softwareentwicklung zu forcieren.

In aktuellen Webanwendungen werden Websockets als Technologie eingesetzt. Seit einigen Jahren ist bereits zu beobachten, dass eine Webanwendung mit Websockets in einer so RAM-beschränkten Umgebung im *tomcat* in den Versionen 8, 9 und nun auch 10 unter Last schnell so viele Ressourcen verbraucht, dass der *OOM-Killer (OutOfMemory)* den Java-Prozess beendet. Das betrifft alle drei verfügbaren Java-Versionen.

3.1 Testkonstellation

Um festzustellen, ob das Problem durch einen Programmierfehler im *tomcat* oder der Java-Runtime verursacht wird, oder aber die beschränkten Ressourcen als solche der Grund sind, wird eine kleine Webanwendung deployt. Sie besteht lediglich aus einem WebSocket-Endpunkt und einem kleinen `ContextListener`, der sekundlich den aktuell genutzten Heap, die Anzahl der empfangenen Nachrichten sowie den vom Tomcat und dem Docker belegten Hauptspeicher (`Resident Set Size - RSS`) ins Log schreibt.

Zusätzlich werden zwei *node*-Prozesse auf einem weiteren Host gestartet, die jeweils 2500 Websockets öffnen und sequenziell Nachrichten an den Endpunkt senden und die Antwort abwarten (Echo-Service). Nach 100 Nachrichten wird der WebSocket geschlossen und ein neuer geöffnet, so dass insgesamt durchgängig über vier Stunden ca. 5000 aktive Websockets verbunden sind. Die Experimente wurden zu unterschiedlichen Tageszeiten durchgeführt.

3.2 Erste Beobachtung

In der Standardkonfiguration des *tomcat 10* unter Java SE 17 stellt sich heraus, dass bei 2 GB virtuellem Speicher der OOM-Killer den Java-Prozess innerhalb von 10 Minuten beendet. Bei zugewiesenen 3 GB virtuellem Speicher läuft der Prozess auch nach Stunden noch. Allerdings steigt der Verbrauch auf über 2.5 GB (siehe Abbildung 2).

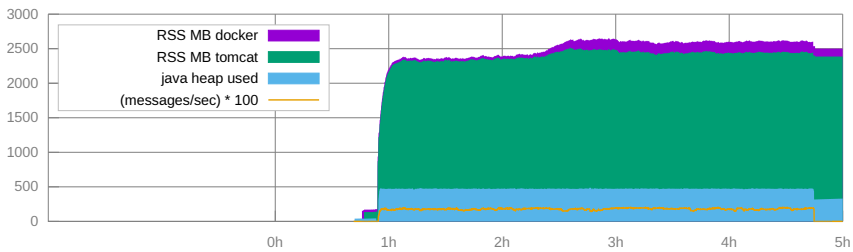


Abb. 2: *tomcat 10* im Docker-Container: der Off-Heap Speicherverbrauch steigt über 2 GB bei maximalem Heap von 512 MB

Zusätzlich ist zu beobachten:

- die Anzahl der Nachrichten pro Sekunde liegt bei etwa 20.000
- der Java-Heap liegt tatsächlich nahezu durchgehend bei 500 MB

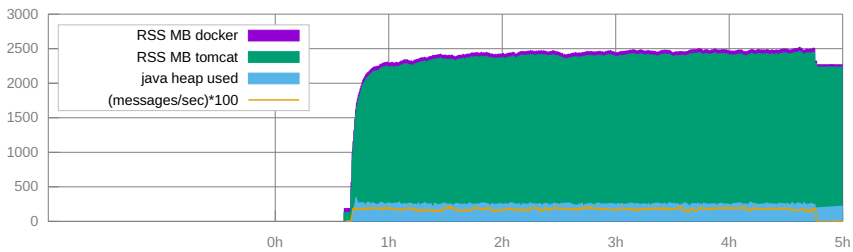
Diese Kombination lässt darauf schließen, dass sowohl auf den Garbage-Collector als auch auf das Verarbeiten der Websocket-Nachrichten nicht unerhebliche Anteile an dem CPU-Verbrauch fallen. In dem vorliegenden Beispiel haben wir den CPU-Verbrauch nicht weiter betrachtet, sondern das Augenmerk auf den Speicherverbrauch gelegt.

Die gleichen Ergebnisse sind mit *tomcat* in den Versionen 8 und 9 sowie den Java-Versionen 9 und 11 zu beobachten. Als Werkzeuge haben sich nach mehreren Experimenten u.a. mit *pmap* und *ps* letztlich *docker stats*, *pidstat* und *jcmd* als besonders geeignet herausgestellt, mit denen die Daten für die Aufbereitungen erzeugt wurden.

Die erste Arbeitshypothese: Irgendwo in der Implementierung des *tomcat* existiert ein Problem, das zu einem übermäßigen Off-Heap Verbrauch führt.

3.3 Prüfung der Hypothese mit *jetty*

Wenn es denn an der Implementierung des *tomcat* liegt, sollte eine andere Servlet-Engine ein anderes Verhalten zeigen. Daher wird die gleiche Anwendung mit den gleichen Parametern (512 MB max Heap, etc.) in *jetty* deployt und mit den gleichen Clients getestet. In Abbildung 3 ist zu erkennen, dass beim *jetty* das gleiche - oder zumindest ein vergleichbares - Problem auftritt: nach sehr kurzer Zeit steigt der von *docker stats* angezeigte Speicherverbrauch auf deutlich über 2 GB und erreicht nach etwas über vier Stunden 2.5 GB. Bekommt der Docker-Container lediglich 2 GB virtuellen Speicher zugewiesen, wird der Java-Prozess nach weniger als 10 Minuten von dem OOM-Killer beendet.

Abb. 3: *jetty 11* im Docker-Container

Weiterhin ist zu beobachten:

- auch hier beträgt die Anzahl der abgearbeiteten Nachrichten pro Sekunde etwa 20.000
- der tatsächlich gebrauchte Heap beträgt im Mittel nur etwa die Hälfte von dem, den der *tomcat* benötigt

Damit ist zwar die Hypothese des Implementierungsproblems im *tomcat* nicht widerlegt, aber da beide Servlet-Engines das gleiche Phänomen zeigen, könnte hier auch ein fundamentales Problem zu Tage treten: es könnte ein generelles Problem in der JVM-Implementierung bzw. in den Standardbibliotheken sein.

3.4 Prüfung der Hypothese mit *wildfly*

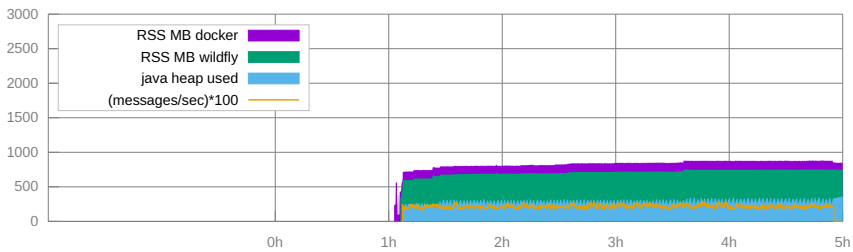
Als weitere Variante wird die Webanwendung im *wildfly* deployt. Interessanterweise zeigt sich hier nicht das Verhalten der beiden anderen Servlet-Engines. Wie in Abbildung 4 zu sehen, steigt der Gesamtverbrauch auf knapp unter 800 MB (und bleibt auch nach zwölf Stunden dort).

Zusätzlich ist zu beobachten:

- die Anzahl der abgearbeiteten Nachrichten pro Sekunde beträgt etwas unter 20.000
- der tatsächlich gebrauchte Heap beträgt im Mittel deutlich weniger als bei *jetty*

3.5 Analyse und zweite Hypothese

Die beiden Servlet-Engines *jetty* und *tomcat* nutzen in der jeweiligen Standardkonfiguration *per-message-deflate* bei Websockets. Bei *wildfly* ist das nicht der Fall.

Abb. 4: *wildfly preview 26* im Docker-Container

Mindestens seit Java SE 8 gibt es offensichtlich häufiger Schwierigkeiten mit Off-Heap Memory Leaks bei Anwendungen, die die Kompressionsalgorithmen aus dem Paket *java.util* nutzen. Augenscheinlich ist die Implementierung recht schwierig, korrekt anzuwenden, so dass die Vermutung naheliegt, dass die Servlet-Engines zur Implementierung des *per-message-deflate*-Features diese Klassen nutzen und es daher zum übermäßigen Speicherverbrauch kommt.

Um die Hypothese zu prüfen, wird das Feature im *tomcat* deaktiviert. Hierbei ist zu beobachten, dass über zwölf Stunden hinweg die Testanwendung im vertretbaren Rahmen unter 1 GB bleibt. Besonders interessant ist, dass die Anzahl der abgearbeiteten Requests pro Sekunde sich mit knapp unter 40.000 fast verdoppelt. Um das Experiment abzuschließen, könnte noch versucht werden, bei *jetty* das Feature zu deaktivieren und es beim *wildfly* zu aktivieren. Beides ist schwierig und mit erheblichem Aufwand verbunden.

3.6 Technisches Zwischenfazit

Für eine produktive Anwendung, in der mit Websockets gearbeitet wird und die über Tage - vielleicht sogar Wochen stabil laufen soll, sind *wildfly* in der Standardkonfiguration und *tomcat* mit deaktiviertem *per-message-deflate* vermutlich gleichermaßen geeignet. *tomcat* stellt sich als leistungsfähiger in Sachen *Messages per Second* heraus, *wildfly* scheint in Bezug auf Heap-Auslastung deutlich effizienter. Hier wären weitere Experimente durchzuführen, um herauszufinden, wie sich die beiden Engines in realeren Anwendungen mit Sessions und mehr fachlichen Objekten verhalten: es mag sein, dass der *tomcat* auf Dauer mehr und mehr Ressourcen auf den Garbage-Collector verwendet und der Heap deutlich vergrößert werden müsste.

In allen Varianten steigt der von *docker stats* angezeigte Speicher sehr langsam aber kontinuierlich an. Keiner der Prozesse im Docker-Container rechtfertigt diese Beobachtung, die beim *tomcat* am deutlichsten auftritt. Zudem - auch wenn es nicht mehr so rapide ist - steigt der Speicher des Java-Prozesses selbst noch immer stetig an. Ein gelegentliches `jcmd $(pidof java) System.trim_native_heap` scheint da etwas Abhilfe zu schaffen.

In jedem Fall gehört diese Art des Systemmonitorings auch in produktiven Systemen zum soliden Handwerk. Ähnliche Beispiele haben die Studierenden in ihren Ausarbeitungen aufgegriffen und eigene Analysen durchgeführt, die am Ende des Semesters in Form einer Plakatausstellung präsentiert wurden.

4 Fazit

Das hier vorgestellte Beispiel des *Off-Heap Memory Leaks in Servlet-Containern* zeigt exemplarisch das Lernsetting in dem Modul *Grundlagen der Qualitätsmanagements* auf. Schritt für Schritt erarbeiten wir gemeinsam mit den Studierenden entlang einer konkreten Implementierung einer Java-Applikation wie eine Analyse eines Gesamtsystems aufgebaut werden kann und wie diese Erkenntnisse in die Verbesserung der Qualität des Gesamtsystems eingehen. In dem konkreten Beispiel konnte durch eine veränderte Konfiguration der Laufzeitumgebung - dem Tomcat - das Problem zumindest teilweise behoben werden ohne zusätzliche Ressourcen dem Container zuzuordnen. Es konnte gezeigt werden, dass eine Analyse des Gesamtsystems auf einfachen, sehr effektiven Werkzeugen des Betriebssystems aufgebaut werden konnte.

Mit weiteren Beispielen haben wir im Laufe der Veranstaltung verschiedene Facetten der Nachhaltigkeit in der Softwareentwicklung aufgegriffen und die Studierenden für das Zusammenspiel von Softwareentwicklung, Qualitätssicherung und den operativen Betrieb sensibilisiert. Durch die gemeinsame Erarbeitung der konkreten Umsetzung der Beispiele in Java-Applikationen hat es für die Studierenden greifbarer gemacht und zumindest teilweise Einzug in ihr Alltagshandeln in der Softwareentwicklung gefunden.

Literaturverzeichnis

- [ERV20] Erb, U.; Radfelder, O.; Vosseberg, K.: Nachhaltigkeitskultur in der Informatik. FifF Kommunikation, 2020(1/20):40–42, 2020.
- [GGP22] Gerstlacher, J.; Groher, I.; Plösch, R.: Green und Sustainable Software im Kontext von Software Qualitätsmodellen. HMD Praxis der Wirtschaft, 59(4):1149–1164, 2022.
- [GTB20] GTB - German Testing Board e.V.: Lehrplan *Certified Tester Foundation Level*, 2020 <https://www.german-testing-board.de> (letzter Aufruf am: 2022-10-20).
- [KS19] Kasteleiner, B.; Schwartz, A.: DevOps - Schnell, zuverlässig und sicher von der Idee zur Realisierung. Informatik Spektrum, 42(3):211–214, 2019.
- [SL19] Spillner, A.; Linz, T.: Basiswissen Softwaretest, 6. Auflage. dpunkt-Verlag, Heidelberg, 2019.
- [WVS21] Winter, M.; Vosseberg, K.; Simon, F.: Technischer Report: Umfrage 2020 - Softwaretest in Praxis und Forschung. 2021, <https://www.softwaretest-umfrage.de/pdf/Technischer-Report-Umfrage-2020-V11Linked.pdf>, (letzter Aufruf am: 2022-10-20).

Session 3

The Microservice Dungeon: Realitätsnahe Lehre komplexer Softwarearchitekturen

Philipp Felix Schmeier,¹ Stefan Bente²

Abstract: Microservices bieten sich als Lösung für große, lose gekoppelte Softwarelandschaften an, die durch weitgehend autonome Teams entwickelt werden. Dies setzt einen geeigneten fachlichen Schnitt, asynchrone Kommunikation und eine Container-orchestrierte Infrastruktur voraus - alles erhebliche Komplexitätstreiber. In der Hochschullehre hat man kaum die Chance, diese Komplexität den Studierenden anschaulich und praxisnah zu vermitteln, da hier aus didaktischen Gründen häufig eher kleinere Software-Einheiten betrachtet werden. Das Verbundprojekt »The Microservice Dungeon« mit 59 Studierenden aus vier verkoppelten Lehrveranstaltungen in drei Informatik-Studiengängen versucht diesen Widerspruch aufzulösen. Die Studierenden konzipieren eine große Microservice-Landschaft und implementieren darin selbst Services, die in einer Spielwelt Roboterschwärme steuern. So entsteht eine Lernplattform für das Software Engineering, die nunmehr in der dritten Iteration weiterentwickelt wird. Anhand detaillierter Feedback-Befragungen und einer Analyse von ca. 3500 Posts im Messenger des Projekts konnte gezeigt werden, dass dieser Ansatz die Studierenden erfolgreich zu einer aktiven Beschäftigung mit Softwarethemen wie REST-Calls, Eventing und DevOps hinführt.

Keywords: Microservice; Lernplattform; Software Engineering; Eventing; REST; lose Kopplung; DevOps; Infrastruktur; Didaktik

1 Einführung

Bei der Vermittlung von Softwarearchitektur-Kompetenzen hat die projektorientierte Hochschullehre mit einem inhärenten Widerspruch zu kämpfen. Einerseits muss die Aufgabenstellung und die entstehende Code-Basis klein genug sein, damit Studierende im Rahmen einer typischen Veranstaltungsgröße - zwischen 5 und 10 ECTS - in einem Semester das Lernziel erreichen können. Andererseits werden sowohl Vorzüge wie auch Mehraufwand einer modernen, cloud-kompatiblen Architektur (mit auf Domain-Driven Design [Ev04] beruhenden Microservices in einer Container-orchestrierten Infrastruktur) erst ab einer gewissen Systemgröße deutlich.

¹ TH Köln, CIDE - Cologne Institute for Digital Ecosystems, Steinermüllerallee 1, 51643 Gummersbach, philipp.schmeier@gmail.com

² s.o., stefan.bente@th-koeln.de

Wenn die Studierenden lernen sollen, hochperformante und skalierbare Systeme wie bei Netflix oder Amazon [Hi21] zu bauen, dann brauchen sie ein spezielles Projektformat mit hinreichender Größe. In der Literatur gibt es hierzu nur wenige dokumentierte Beispiele³.

Aus diesem Grund wurde durch die Autoren ein großes Verbundprojekt mit insgesamt 59 Studierenden in vier integrierten Lehrveranstaltungen konzipiert, das eine praxisnahe Microservice-Architektur umsetzt. Ein wesentliches Ziel von Microservices ist es, Teams ein weitgehend autarkes Arbeiten ohne dauernde Abstimmungen über die Teamgrenzen hinweg zu ermöglichen. Das Verbundprojekt war mit der großen Menge an teilnehmenden Studierenden bewusst darauf angelegt, ein im Studium sonst nicht übliches Maß an Komplexität zu schaffen. Die Studierenden sollten den »Schmerz« dauerhafter Abstimmungsprozesse erfahren. Daraus, so die Intention der Lehrveranstaltung, sollte sich die Motivation speisen, in Tests, Dokumentation und geeignete Architekturen (Redundanz, asynchroner statt synchroner Nachrichtenaustausch, etc.) zu investieren, um Abstimmungsbedarfe zu reduzieren und ungestörter arbeiten zu können. Mit anderen Worten: Das Projekt hatte den Anspruch, reale Berufsbedingungen in puncto Komplexität zu simulieren.

Gleichzeitig sollte das Projekt die Studierenden ermuntern, fachliche und technische Schwierigkeiten zu diskutieren und im Konsens zu Lösungen zu kommen. Im Sinne echter »self-organized teams« wurde seitens der Betreuer bewusst darauf verzichtet, zu viele Vorgaben im Detail zu machen. Dadurch sollte den Studierenden Raum für die eigene Lösungsfindung geschaffen werden. Darüber hinaus war es das Ziel, alle Teilnehmenden möglichst gleichermaßen zu beteiligen (keine »Trittbrettfahrer«). Kompetenzen sollten mindestens auf Stufe 3 (*Applying*), besser 5 (*Evaluating*) der Revised Bloom's Taxonomy [AK01] erworben werden. Das Verbundprojekt wurde im Rahmen einer Masterarbeit [Sc22] empirisch begleitet, auf deren Ergebnissen das vorliegende Paper zu großen Teilen beruht.

2 Aufbau und Phasen des Verbundprojekts

Unter dem Titel »The Microservice Dungeon« wurden für das Projekt im Wintersemester 21/22 vier Lehrveranstaltungen aus drei Informatik-Studiengängen zusammengekoppelt. Die Entwicklung eines Spiels wurde gewählt, um den Studierenden eine interessante und ihrer Lebenswirklichkeit nahestehenden Domäne zu bieten. Dies half erkennbar bei Diskussionen über Fachlichkeit - so gut wie alle Beteiligten waren »Gamer« und konnten Erfahrung zu bestimmten Mustern und Regelsystemen beisteuern [Sc22].

Das Verbundprojekt befindet sich zum Zeitpunkt, an dem dieses Paper geschrieben wurde, bereits in der dritten Iteration. Im Folgenden wird aber nur die erste Projektiteration

³ Eine erwähnenswerte Ausnahme ist das »Laboratory of Complex Computational Systems« an der University of São Paulo (USP), bei dem eine Microservice-Landschaft mit einer großen Gruppe von Studierenden durchgeführt wurde (4-mal als zweiwöchiger Kurs mit insgesamt 65 Studierenden und einmalig einsemestrig mit 18 Studierenden) [Co20]. Das aus dem Projekt entstandene Paper fokussiert allerdings eher auf erreichte »Hard Skills« und weniger auf eine detaillierte Analyse von Projektformat und Aufgabenstellung aus didaktischer Sicht.

beschrieben und analysiert, da es sich bei dieser um die zahlenmäßig größte Version des »Microservice Dungeon« handelt und sie damit die besten Möglichkeiten für Analysen liefert. Die nachfolgenden Iterationen werden in Kap. 5 kurz umrissen.

Das Projekt wurde vom 13.09.2021 bis zum 28.01.2022 durchgeführt. Die verschiedenen Module hatten teilweise unterschiedlichen Aufgaben und Rollenzuweisungen. Die Teilnehmenden des einzigen Master-Moduls »Domain Driven Design of large Software Systems« übernahmen beispielsweise die Rolle von Architekt:innen und Product Ownern. Abb. 1 zeigt eine Übersicht über alle Teilnehmenden und deren Rollen.

Kürzel	Studiengang	Module	ECTS	Personenzahl	Rollen	DDD	MSA	PL	IP
DDD	Digital Sciences (Master)	Domain Driven Design of large Software Systems	6	10	Architekt:in Product Owner	x x			
MSA	Code and Context (Bachelor)	Microservices Architecture	3	29	Gameplay Designer			x	
PL	Code and Context (Bachelor)	Project Launch	12	5	Software Developer (Service)	(x)		x	x
IP	Informatik (Bachelor)	Informatikprojekt	10	15	Software Developer (Player)	(x)	x	x	x

Abb. 1: Übersicht der Teilnehmenden

Das Projekt begann als Greenfield-Projekt und ermöglichte dadurch den Studierenden große Freiräume bei der Gestaltung, von der Systemlandschaft bis hin zu den verwendeten Technologien. Die Studierenden hatten weitgehende Technologiefreiheit, was auch als Vorteil von Microservice-Systemlandschaften gilt [Ne15]. Dies führte zu einer gewissen Technologie-Vielfalt in den fünf Core-Services.

Die Lehrenden beschränkten die eigene Einflussnahme weitgehend und gaben nur Rahmen und Ziele vor. Im Vordergrund stand die Moderation von Veranstaltungen und Statustreffen. Die genauen Inhalte und Regeln des entstehenden Spiels wurde den Studierenden, genauer der Subgruppe der »Gameplay Designer/Dungeon Master« (Gruppe PL in Abb. 1) überlassen, die über das gesamte Projekt hinweg als Ansprechpartner und Expert:innen für die Spielregeln agierten. Der Ablauf des Projekts wurde in verschiedene Phasen unterteilt, wie in nachfolgender Abbildung dargestellt.

KW	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	1	2	3	4
	B		K		Development												PS		P	
					CS															

Bewerbung, Konzeption, Core Services, Player Services, Präsentation

Abb. 2: Projektphasen nach Kalenderwoche (2021/22)

Bei drei der vier teilnehmenden Module handelte es sich um Wahlpflicht-Module, nur MSA ist ein Pflichtmodul. Bei Modul IP wurde eine Teilnehmer-Obergrenze von 15 festgelegt. Die Bewerber:innen hatten die Möglichkeit, mittels Bewerbungsvideos ihre Motivation für die Projektthematik darzustellen, was als Auswahlkriterium diente. Daher kann von einem gewissen intrinsischem Interesse der teilnehmenden Studierenden an dem Thema ausgegangen werden.

2.1 Konzeption

Die Gameplay Designer (PL) hatten vor Beginn Zeit, die Spielregeln festzulegen, die im Semester umgesetzt werden sollten. Bis auf wenige Eingriffe der Lehrenden (etwa um den Umfang der Spielmechaniken zu reduzieren) stand es den Designern frei, welche Form die Regeln annehmen sollten. Die Phase wurde mit einer Präsentation des geplanten Gameplays für alle⁴ Projektteilnehmenden abgeschlossen. Ebenfalls in diesem Rahmen wurde ein *Event Storming* durchgeführt, dessen Ergebnisse in Abb. 3 dargestellt sind.



Abb. 3: Übersicht der Ergebnisse des Event Storming Workshops

Die von Alberto Brandolini entwickelte Methodik des Event Storming ermöglicht die explorative Erforschung einer komplexen Fachdomäne. Ursprünglich mit dem Hintergrund der Softwareentwicklung entworfen, bietet dieses Vorgehen Einsatzmöglichkeiten bis hin zu Organisationsentwicklung [Br21]. Für das Projekt diente das zweitägige Event Storming mehreren Zwecken. Einerseits wurde die Fachlichkeit (Spielablauf und Regeln) von den Studierenden durchdrungen, hinterfragt und verinnerlicht. Andererseits konnten aus den Ergebnissen heraus die Domänengrenzen der Core-Services identifiziert werden.

Durch das interaktive Format konnte der Herausforderung durch die große Zahl und Diversität der Teilnehmenden begegnet werden. Besonders angesichts der ansonsten aufgrund der Pandemiesituation häufigen Remoteveranstaltungen wurde das Präsenzformat als lebendig und inspirierend wahrgenommen, und bot allen Beteiligten die Möglichkeit, sich persönlich kennenzulernen. Besonders aus didaktischer Sicht kann vermutet werden, dass dieses Treffen zu einem frühen Projektzeitpunkt zu der positiven, lebendigen Stimmung in den Projektdiskussionen (siehe Kap. 4) beigetragen hat.

Die konzipierten und durch das Event Storming »ausgereiften« Spielregeln wurden für alle Teilnehmenden öffentlich einsehbar über eine Website dokumentiert [Mia]. Die Spiellandschaft stellt eine zweidimensionale Karte dar, die den teilnehmenden Playern unbekannt ist. Jedes Feld der Karte spiegelt einen Planeten wider, der verschiedene Ressourcen enthalten kann. Ein Player ist kein Mensch, sondern ein autonom agierender und in der Infrastruktur deployter Service. Hinter einem Player steht ein Studienteam, das ihn implementiert.

Teilnehmende Player erhalten ein Startguthaben und können sich damit erste Roboter kaufen, die auf definierten Stellen auf der Karte starten. Die Player kennen zu Beginn nur die Planeten, auf denen ihr Roboter startet. Ziel der Player ist es, durch geschicktes Erkunden,

⁴ MSA war nicht Teil der Einführungsveranstaltung, da MSA in einer Blockveranstaltung gelehrt wird. Die MSA-Studierenden stießen daher erst gegen Ende des Projektes (Playerentwicklung) dazu.

Abbauen und Verkaufen von Ressourcen sowie durch Kämpfe einen Roboterschwarm aufzubauen und Punkte zu sammeln, die für jede Interaktion im Spiel verteilt werden.

2.2 Development

Die Gestaltung der Spielregeln, also der Karte, der Überwachung der Roboterbewegung, der Anmeldung zum Spiel und die Verwaltung der Guthaben der Spieler war die erste Aufgabe für die Studierenden. Damit entstanden fünf »Core-Services«. Die Entwicklung der eigenen Robotersteuerung und Strategien wurde zeitlich und thematisch davon abgegrenzt, dies sind die »Player-Services«. Diese Abgrenzung hatte im Wesentlichen drei Gründe:

1. Alle Teilnehmenden, die sowohl Core- wie auch Player-Services entwickeln, lernten durch die Wiederholung, essenziellen Aspekte noch besser zu verinnerlichen.
2. Alle Teilnehmenden mussten gegen vorgegebene REST- und Event-Spezifikation entwickeln (für die Player-Services), wie auch solche Spezifikationen selbst entwickeln (Core-Services).
3. Teilnehmende aus dem MSA-Modul, die aus Lehrplan-Gründen erst spät hinzukommen konnten, konnten zumindest noch in der Player-Entwicklung dabei sein.

Die Entwicklung der Systemlandschaft mit den Core-Services stellte die zeitlich längste Phase des Projektes dar. Die im Serviceschnitt im Event Storming entstandenen Services wurden Studierendengruppen von 2 bis 5 Softwareentwickler:innen zugeordnet. Jedem so entstandenen Development Team aus Bachelor-Studierenden wurden zwei Master-Studierende als Product Owner und Architekt:in zugeordnet. Über einen Zeitraum von 12 Wochen entstand in einer durch agiles Vorgehen inspirierten Projektstruktur die Servicelandschaft. Die entstandenen Services können in GitHub [Mic] (1. Iteration) und GitLab [Mib] (aktueller Stand) eingesehen werden. An Scrum orientiert wurden 2-wöchentliche Sprint-Review- und -Planning-Meetings abgehalten, die durch die Product Owner geleitet wurden. Lehrende begleiteten alle Meetings als Beobachter und Antwortgeber für Fragen zu Prozessen oder organisatorischen Themen⁵.

Nach Fertigstellung der Core-Services wurde die Playerentwicklung angestoßen, die über zwei Wochen im Januar stattfand. Dieser Prozess konnte aufgrund zeitlicher Verzögerungen und Entwicklung der Pandemie nicht wie geplant als durchgängiger einwöchiger Hackathon am Campus stattfinden. Der Abschluss der Playerentwicklung gipfelte in einem *Codefight*, in welchem die Player-Services gegeneinander antraten.

Das Projekt wurde mit Kurzpräsentationen der Studierenden abgeschlossen. Aufgabe war die Reflektion des eigenen Handelns im Projekt. Die Themen konnten dabei von den

⁵ Die agile Arbeitsweise war kein definiertes Lernziel der Veranstaltung, um die Veranstaltung nicht zu überfrachten. Die gewählte Form hat sich aber in zahlreichen Lehrforschungsprojekten der Autoren bewährt.

Studierenden frei gewählt werden, sofern es einen Bezug zum Projekt gab. Von Ideen und Konzepten für eine weitere Projektiteration über technische Aspekte bis hin zu Darstellung der eigenen Arbeitsweise im Team war das Ergebnis breit gefächert. Dies war einer von mehreren möglichen Feedbackkanälen für die Studierenden. Weitere werden in den folgenden Kapiteln näher erläutert und analysiert.

3 Feedback der Studierenden

Wie bei vielen realen Software-Projekten traten auch im »Microservice Dungeon« Herausforderungen aller Art auf, von Schwierigkeiten im eigenen Team über unterschiedliche Code-Qualität bis hin zur projektweiten Kommunikation. Um den Projekterfolg und die Vermittlung der Learning Outcomes zu prüfen sowie Hinweise für das Format zu sammeln, wurden verschiedene Feedbackschleifen bereitgestellt, die es den Studierenden ermöglichten, Vorschläge, Kritik und Wünsche zu äußern.

Die erste und dauerhafte Möglichkeit bestand im direkten Feedback, entweder nur an die Lehrenden oder an das gesamte Projektteam. Für den Großteil der Kommunikation wurde die Plattform Discord⁶ verwendet, die ein Instant-Messaging sowohl als Privatschats als auch über Gruppenkanäle ermöglicht. Eine dedizierte Auswertung der über Discord stattgefundenen Kommunikation ist in Kap. 4 zu finden. Zusätzlich dazu wurden zwei Umfragen durchgeführt, eine Zwischenumfrage zur Mitte des Projektes sowie die Abschlussumfrage unmittelbar nach Abschluss der Projektarbeiten. Zuletzt konnten die Studierenden noch durch die Abschlusspräsentation mit ihrer weitgehend freien Themenwahl Vorschläge einbringen und Herausforderungen beleuchten. Ergänzend lieferte ein Teil der Masterstudierenden eine ausführliche individuelle Reflektion des Projekts als Wahlleistung.

Das Feedback durch die Abschlussumfrage fiel sehr positiv aus, insbesondere mit Blick auf erlernte Hard-Skills. So gaben über zwei Drittel der Teilnehmenden an, ein wenig oder sogar viel besser im Programmieren geworden zu sein [Sc22]. Dies ist besonders bemerkenswert, da seitens der Lehrenden aus Zeitgründen kaum Hilfestellung zu konkreten Programmieraufgaben gegeben werden konnte. Weiterhin ist die Verständlichkeit von Microservices im generellen sowie die Nachvollziehbarkeit von Vorteilen und Bedeutung von Microservices fast durchgängig bei allen Teilnehmenden stark gestiegen, wie in Abb. 4 zu sehen⁷.

⁶ Discord ist ein Onlinedienst für Kommunikation und hat ihren Ursprung im Bereich des Gamings und ermöglicht die Kommunikation via Textkanälen, Sprachkonferenzen (inkl. Screenshare und Webcam) und Instant Messaging.

⁷ Das Feedback wird aus Platzgründen hier zusammenfassend dargestellt. Für eine weiter ausdifferenziertere Analyse sei auf [Sc22] verwiesen.

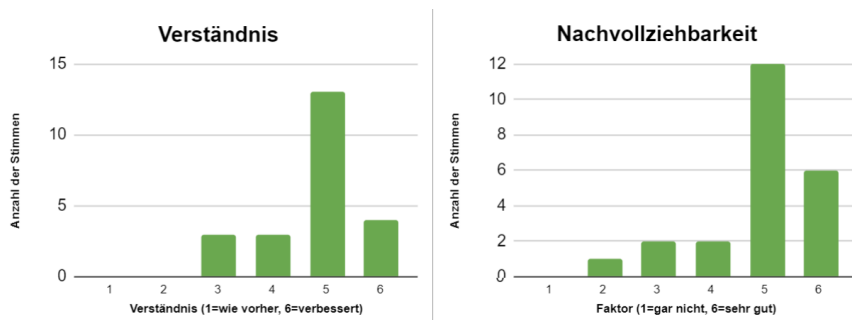


Abb. 4: Verständnis und Nachvollziehbarkeit von Microservices und ihren Vorteilen [Sc22]

Die Erwartungen an das im Projekt Gelernte wurden für einen großen Teil der Studierenden deutlich übertroffen, wie Abb. 5 (links) zeigt. Die meisten Teilnehmenden gaben an, mehr (Stufe 4) oder deutlich mehr (Stufe 5) gelernt zu haben als erwartet. Neu erworbene Kompetenzen konnten dabei auch wirksam in der Implementierung der Core-Services genutzt werden. Dadurch war das eigene Qualitätsempfinden bei den Core-Services allgemein sehr hoch (Abb. 5, rechts).

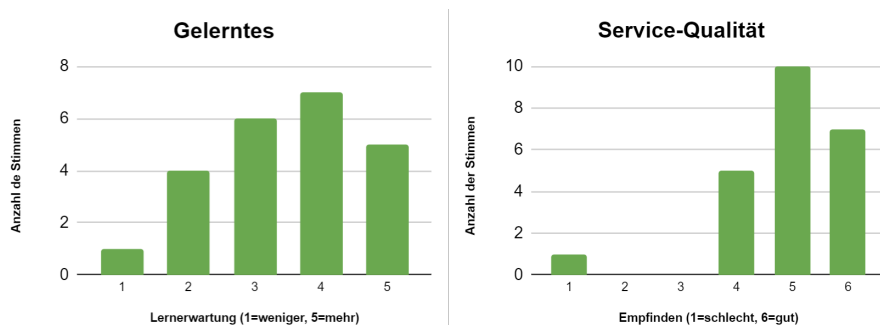


Abb. 5: Übersicht der studentischen Selbsteinschätzung bzgl. Lernerwartung und der produzierten Code-Qualität [Sc22]

Trotz verschiedener Probleme und Herausforderungen in Abstimmung, Arbeitsverteilung und anderen Aspekten wurde das Projekt allgemein als großer Erfolg wahrgenommen. Dies gilt aus Sicht der Studierenden sowohl für die Freude an der Projektarbeit als auch für das Endergebnis, wie Abb. 6 zeigt. Die Spaßtreiber sind dabei breit aufgestellt, die Studierenden nannten beispielsweise das »Event Storming«, »die fachlichen Diskussionen«, das »Entwickeln des eigenen Microservices« oder den Moment, »wenn die Services angefangen haben miteinander zu interagieren« als Höhepunkte der Projekts [Sc22].

Besonders die Zufriedenheit mit dem Projektergebnis ist hervorzuheben. Sie ist um so bemerkenswerter, da der gegen Ende des Projektes geplante Hackathon ausfallen musste und die entwickelten Player-Services angesichts der Projektverzögerungen nicht so weit ausgereift waren, um wirklich spannende Codefights zu ermöglichen. So wurde auch die Projektgröße

als etwas überdimensioniert wahrgenommen. Obwohl also das nominelle, zu Beginn ausgegebene Projektziel »Wir machen einen großen Codefight« nur sehr eingeschränkt erreicht werden konnten, nahmen die Studierenden das Projekt trotzdem als Erfolg wahr. Aus Lehrendensicht war das Betreuungsaufwand durchaus hoch, aber nicht höher als die Summe der Aufwände bei einer »normalen« Durchführung als isolierte Einzelveranstaltungen.

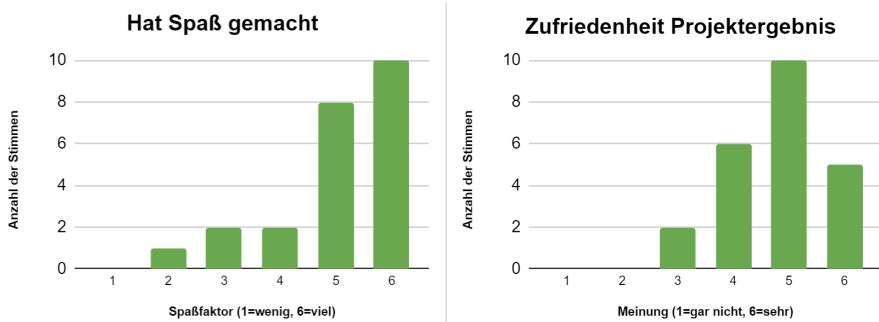


Abb. 6: Spaßfaktor und Zufriedenheit der Studierenden mit dem Projektergebnis [Sc22]

Die Umfrageergebnisse als Ganzes bestätigen den Lernerfolg der Studierenden, besonders im Bereich der Hard-Skills. Fortschritte und Lernerfolge in Reflektionsfähigkeit, Kommunikationsverhalten und Architektur-Knowhow war in den Umfragen kaum greifbar zu gestalten und lässt sich daher nicht in belastbaren Zahlen darstellen. Sie waren in den Studierenden-Präsentationen zur Abschlussreflektion dennoch ablesbar, die ein durchgehend hohes Niveau hatten.

4 Kommunikationsanalyse

Im Projekt wurde Discord als Messenger-Dienst verwendet. Dafür wurden acht dedizierte Kanäle angelegt (je einer pro Core-Service, einer für DevOps, einer für Player-Entwicklung und einer für allgemeine Fragen). Da Discord kein offizielles Tool der Hochschule ist, wurde die Kommunikation via Discord den Studierenden nicht vorgeschrieben. Es wurde dennoch rege genutzt, da davon ausgegangen werden kann, dass Discord durch die Nähe zur Gaming-Community einem hohen Prozentsatz der Teilnehmenden vertraut ist. Viele der Studierenden kennen es auch aus ihrem 3. und 4. Bachelor-Semester, wo es im Modul Softwaretechnik erfolgreich als Diskussionsforum eingesetzt wird [BIK22].

Insgesamt wurden in der Laufzeit des Projekts 3485 Posts abgesetzt, das entspricht einem Mittel von ca. 25 am Tag. In der Spitze war es die zehnfache Zahl, 252 (wenig überraschend: am letzten Tag vor dem Codefight). Diese Posts wurden für dieses Paper aus Discord exportiert, thematisch codiert und ausgewertet, um besseren Einblick in die diskutierten Themen und die Beteiligung der Studierenden zu erhalten.

4.1 Thematische Verteilung

Die inhaltliche Lebendigkeit und Vielfalt der Diskussion lässt sich an Abb. 7 ablesen. Auf der X-Achse sind jeweils die Projektwochen aufgetragen, auf der Y-Achse die Anzahl Posts pro Woche in einer bestimmten thematischen Ausprägung. Discord wurde keineswegs nur für organisatorische Themen wie etwa Terminabsprachen genutzt (links, gestrichelte Linie). Dies machte nur zu Beginn, in einer "Findungsphase", die Mehrzahl der Posts aus. Gegen Ende, als es auf die Reflektions-Präsentation mit zugehöriger Benotung zuging, nahm die Anzahl der nicht-inhaltlichen Posts wieder leicht zu. Es überwogen aber ab etwa der Mitte des Projekts die inhaltlichen Themen.

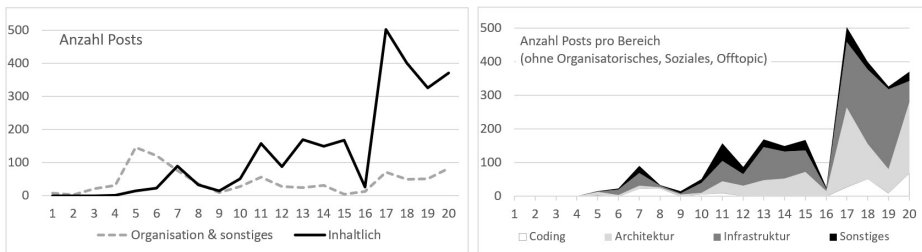


Abb. 7: Verlauf der Posts nach thematischen Bereichen (»Knick« bei Woche 16 = Weihnachtswoche)

Dabei dominierten überraschenderweise nicht unbedingt die reinen »Coding«-Themen, die das Programmieren im Kleinen (mit den damit einhergehenden Verständnisfragen) abdecken (rechts, unterstes weißes Band). Hier scheinen die Studierenden die eingesetzten Technologien gut zu beherrschen. Stattdessen wurde außerordentlich viel über Architektur (hellgrau) und Infrastruktur (dunkelgrau) diskutiert - ein erfreuliches Zeichen, dass die Studierenden offensichtlich eines der Projektziele aktiv angenommen hatten, nämlich sich mit der Tauglichkeit von Architektur- und Hostingansätzen auseinanderzusetzen.

Schaut man sich die Schwerpunkte bei Architektur-Diskussionen genauer an (Abb. 8 links), so werden die Themen »Eventing« (Struktur und Payload der auf Kafka-Topics veröffentlichten Events) und »REST-API« (Aufbau und Implementierung der REST-Endpunkte) mit annähernd gleichbleibender Intensität von der Mitte des Projekts an bis zum Ende diskutiert. Die Intensität beim Thema »Choreographie« (dunkelgraues Band: Welche Event-Konsumierungen und synchronen Aufrufe müssen in welcher Reihenfolge erfolgen, um gewisse Transaktionen zu ermöglichen?) nimmt hingegen zum Ende des Projekts hin zu. Dies korreliert mit der Phase der Player-Entwicklung (Phase PS in Abb. 2), wo sich die Player-Services der Fertigstellung nähern und sich Integrationsprobleme beim komplexen Synchron-Asynchron-Zusammenspiel zeigen.

Bei den Infrastruktur-Themen (Abb. 8 rechts) nimmt das Thema »CI/CD« (Anbindung des Service an das Deployment) in der Projektmitte den größten Raum ein, wo alle Services in einer ersten »Hello-World«-Ausbaustufe vorlagen. Die Infrastruktur selbst (Aufbau der

Serverlandschaft, Zuteilung von Ressourcen, Monitoring) blieben eher ein Randthema bis ganz zum Schluss, als die Integrationsphase kurz vor dem Codefight die bislang unentdeckten Probleme aufzeigte.

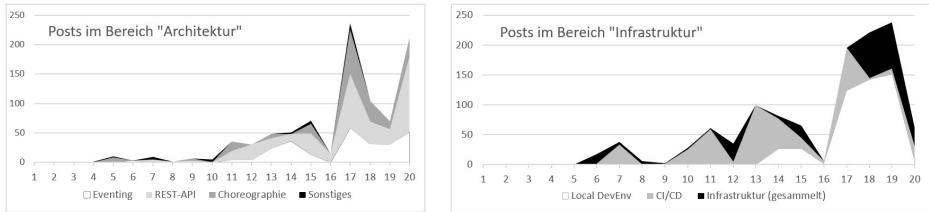


Abb. 8: Verlauf der Posts in den Bereichen »Architektur« und »Infrastruktur«

Das Thema »Local DevEnv« (weißes Band in Abb. 8 rechts) dominiert - auf den ersten Blick überraschend - die Endphase des Projekts. Dies zeigt ein Problem bei der Entwicklung verteilter Systeme auf: Um den eigenen Service lokal testen zu können, muss ein ungleich höheren Aufwand betrieben werden als bei monolithischer Entwicklung (da ja für einen lokal laufenden Player-Service alle Core-Services ansprechbar sein müssen). Dies wurde im Projekt durch eine spezielle, auf Docker-Images und Scripten beruhende lokale Umgebung ermöglicht, in der die fünf Core-Services zur Verfügung gestellt wurden.

Diese Umgebung führte immer wieder zu Nachfragen, weil die Studierenden sich mit einer Menge an komplexer Technologie auseinandersetzen mussten. Das wurde aber erst in dem Moment auf breiter Front relevant, als die Player-Services aus der Unit-Test-Phase in die Integration übergingen. Dies unterstreicht anschaulich, dass das Verbundprojekt seinen Praxisnähe-Anspruch gut einlöst: Die Studierenden stoßen auf dieselben Probleme, von denen auch Microservice-basierte Produktivprojekte aus der »realen Welt« berichten.

4.2 Diskussionsauslöser

Auslöser für Diskussionen waren häufig konzeptionelle Fragen, aber oft auch Verständnisprobleme und Hinweise auf Probleme (Laufzeitfehler, HTTP-Fehlercodes als Returnwerte von REST-Calls, Fehler beim Konsumieren von Events, Inkonsistenzen zwischen Dokumentation und Implementation, etc.). Diese zweite Art von Auslösern wurden in der Codierung der Posts mit einem pauschalen »Problem«-Flag berücksichtigt.

Man erkennt in Abb. 9 links, dass bei den Architekturthemen (schwarze Linien) die Problemgetriggerten Diskussionen erst gegen Ende in der Integrationsphase einsetzen. Zuerst werden Probleme mit der synchronen Kommunikation via REST evident (schwarze gestrichelte Linie) - vermutlich, weil diese aus den Veranstaltungen im 4. Semester besser bekannt ist [Be22]. Die Probleme mit asynchroner Kommunikation (schwarze durchgezogene Linie) treten erst später zutage. Wie oben in Abb. 8 zu erkennen ist, beginnt die konzeptionelle

Diskussion zu diesen Themen deutlich früher, nur die aktive Inanspruchnahme durch Player-Services wird durch die »Fehler-Peaks« markiert.

Auch im Bereich Infrastruktur sind die »Ausschläge« mit hohem Anteil an Problem-Auslösern konsistent mit dem Zeitpunkt, an dem das Verbundprojekt das jeweilige Thema in die Breite brachte. Pipelines für Core-Services und erste, halbfertige Player-Services wurden vom DevOps-Team schon relativ früh im Projekt vorbereitet, daher treten auch die ersten Probleme im Bereich CI/CD (graue gestrichelte Linie) schon früh auf. Probleme in der Infrastruktur kommen erst kurz vor Produktivsetzung bei den Testläufen zum Codefight ans Licht, als das Zusammenspiel aller Komponenten erprobt wurde. Auch hier verhält sich das Verbundprojekt sich »praxisnah« und zeigt ähnliche Muster wie bei Realprojekten.

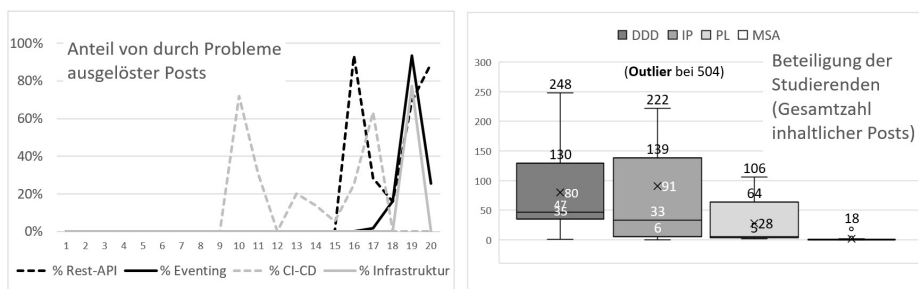


Abb. 9: Anteil der durch Probleme ausgelösten Posts in den Bereichen (links) und Beteiligung der Studierenden (rechts)

4.3 Beteiligung und Diversität

Die bisherige Analyse belegt, dass die Diskussionskultur in dem Verbundprojekt als lebendig, fachlich geprägt und praxisnah angesehen werden kann. Rechts in Abb. 9 sieht man einen Box-Plot der Beteiligung (gesamte Zahl Posts pro Teilnehmendem, aufgeschlüsselt nach Kurs). Sieben Teilnehmende (IP(BA): 4, PL(BA): 1, DDD(MA): 2), steuerten jeweils mehr als 4% der gesamten Menge von 2590 inhaltlichen Posts bei, also jeweils mehr als 100 Posts. Ein Teilnehmer aus dem Informatikprojekt stellte mit 504 inhaltlichen Posts ca. 20% des Gesamtvolumens. Es war immer wieder festzustellen, dass diese *Student Leaders* [AJ22] die allgemeine inhaltliche Diskussion anstießen, bei Problemen halfen und dadurch ein lebendiges Interaktionsklima entstehen ließen.

Die Diskussion fand aber durchaus auch in der Breite der Teilnehmerschaft statt. Der Master-Kurs DDD und das Informatikprojekt IP haben hier ein vergleichbares Profil. Median und oberes/unteres Quartil liegen bei ähnlichen Werten, wobei der Master-Kurs homogener ist. Das aus fünf Teilnehmenden bestehende PL hat ein sehr aktives Mitglied und vier weitere, die sich in Workshops durchaus beteiligten, aber kaum an der Fachdiskussion teilnehmen. Insgesamt haben knapp 2/3 der Teilnehmenden aus DDD, IP und PL (18 von 30) mindestens

30 inhaltliche Posts verfasst. Bei den nur abschnittsweise teilnehmenden MSA-Studierenden haben sich immerhin noch knapp 1/3 (9 von 29) mindestens ein Mal zu Wort gemeldet.

5 Fazit und Ausblick

Ziel des Verbundprojekts war es, den Studierenden über ein breit angelegtes Projekt die Möglichkeit zu geben, die Entwicklung einer Microservice-Architektur in einem realitätsnahem Rahmen aktiv mitzugestalten. Dieses Ziel kann nach der Analyse des Feedbacks als erreicht betrachtet werden. Die Analyse der Projektkommunikation zeigt darüber hinaus eine hohe Aktivierung der Studierenden für die erfolgreiche Auseinandersetzung mit Architektur- und Infrastrukturthemen.

Dennoch wurde durch große Teile des Feedbacks auch Vieles aufgezeigt, das in weiteren Iterationen zu verbessern wäre. Der größte Vorteil der weiteren Projektiterationen liegt in der Tatsache, dass durch die Vorgängersemester bereits eine Systemlandschaft bereitsteht, die verstanden, erweitert und genutzt werden kann.

Die zweite und dritte Iteration im SoSe 2022 und WiSe 2022/23 wurde in etwas kleinerem Rahmen mit je zwei Modulen durchgeführt. Der Fokus lag in der zweiten Iteration zuerst auf einer Analyse und Weiterentwicklung, besonders hinsichtlich Robustheit der Servicelandschaft. In der dritten Iteration lag das Ziel auf einem Refactoring hin zu einer echten *Event-Driven Architecture* (EDA) der Systemlandschaft. Dabei fokussierten sich die Bachelorstudierenden auf die Entwicklung von Player-Services, ihre Master-Komponenten auf Core-Services. Besonders die Player-Entwicklung ermöglicht es, auch in zukünftigen Iterationen stetig neue Aspekte für die Studierenden zu schaffen. Weiterhin lassen sich auch Spielregeln und damit einhergehend Core-Services erweitern, so dass stetig neue Anreize und Herausforderungen für die Studierenden geschaffen werden können.

Der aktuelle Stand des Projekts ist unter [Mia] dokumentiert. Alle Sourcen sind öffentlich unter [Mib] zugänglich. Interessierte Lehrende sind herzlich eingeladen, das Konzept für die eigene Lehre zu adaptieren. Dabei ist sowohl ein umfassendes Lehrforschungsprojekt wie hier beschrieben denkbar, wie auch die Nutzung als fertige »Sandbox« für studentische Experimente mit Microservice- und EDA-Ansätzen.

An der TH Köln ist ein Forschungsantrag in Arbeit, um die Sourcen weiter zu entwickeln und das Projektformat damit auch langfristig bereitstellen zu können. Für eine Pflichtveranstaltung ist das Format vermutlich weniger geeignet, da wenig Fokus auf Prüfungsdesign liegt. Als Wahlveranstaltung für an Coding und Architektur interessierte Studierende erreicht es aber nachgewiesenermaßen die große Mehrheit der Teilnehmenden, und ist damit eine große Bereicherung des Curriculums.

Literaturverzeichnis

- [AJ22] Al-Jarf, Reima: Interaction Analysis in Online Learning Communities: The Student Leader. *Journal of Learning and Development Studies*, 2(2):22–38, Jul. 2022.
- [AK01] Anderson, Lorin W.; Krathwohl, David R., Hrsg. *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. Allyn & Bacon, New York, 2. Auflage, December 2001.
- [Be22] Bente, Stefan: , Learning Outcome der Veranstaltung Softwaretechnik 2 (ST2), 3 2022. Abgerufen 31.10.2022.
- [BIK22] Bente, Stefan; Intveen, Jann; Krampe, Fabian: Divekit — Digitalisierung und Individualisierung als Schlüssel für eine moderne Softwaretechnik-Ausbildung. In (Thurner, Veronika; Kleinen, Barne; Siegeris, Juliane; Weber-Wulff, Debora, Hrsg.): *Software Engineering im Unterricht der Hochschulen (SEUH 2022)*. Gesellschaft für Informatik, Bonn, S. 29–41, 2022.
- [Br21] Brandolini, Alberto: *Introducing EventStorming: An act of Deliberate Collective Learning*. Leanpub, 2021.
- [Co20] Cordeiro, Renato; Rosa, Thatiane; Goldman, Alfredo; Guerra, Eduardo: *Teaching Complex Systems based on Microservices*. GROUP, 1:1, 2020.
- [Ev04] Evans, Eric: *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
- [Hi21] Hillpot, Jeremy: , 4 Microservices Examples: Amazon, Netflix, Uber, and Etsy. <https://blog.dreamfactory.com/microservices-examples>, 2021. Abgerufen 25.01.2022.
- [Mia] *Microservice Dungeon: Aktuelle Dokumentation*. <https://the-microservice-dungeon.gitlab.io/docs>. Abgerufen 05.11.2022.
- [Mib] *Microservice Dungeon: Git-Repositories*. <https://gitlab.com/the-microservice-dungeon>. Abgerufen 05.11.2022.
- [Mic] *Microservice Dungeon: GitHub-Repositories (1. Projektiteration)*. <https://github.com/The-Microservice-Dungeon>. Abgerufen 08.01.2023.
- [Ne15] Newman, Sam: *Microservices: Konzeption und design*. MITP-Verlags GmbH & Co. KG, 2015.
- [Sc22] Schmeier, Philipp Felix: *Moderne lose gekoppelte Softwarearchitekturen: Wie die Hochschullehre komplexe Systeme „hands-on“ vermitteln kann*. Masterarbeit, Technische Hochschule Köln, Steinmüllerallee 1 51643 Gummersbach, Februar 2022.

OPPSEE – eine Online-Plattform zum Programmieren Üben

Axel Schmolitzky,¹ Henri Burau²

Abstract: Programmieren Können ist eine Kernkompetenz für Informatiker*innen. Um gut Programmieren zu können, ist sehr viel Programmieren Üben notwendig. Es gibt etliche Online-Plattformen, auf denen Programmieraufgaben zum Üben zur Verfügung gestellt werden. Ein Problem für Lernende auf diesen Plattformen ist häufig, die zum eigenen Lernfortschritt passenden Aufgaben zu finden. Wir stellen eine neue Plattform mit einem klaren Fokus auf das Programmieren Üben vor und beschreiben ihre flexible Architektur.

Keywords: Programmieren; Programmierlehre; Softwareentwicklung; Automatisierte Bewertung von Software; Programmierplattformen

1 Einleitung

Programmieren Können erfordert, neben weiteren Fähigkeiten, sowohl Lese- als auch Schreibkompetenz für formalisierte Texte (Quelltexte) und damit einige der Kernkompetenzen für ein erfolgreiches Informatikstudium [Be21]. Für viele Studierende der Informatik an der HAW Hamburg sind die einführenden Programmierveranstaltungen die höchsten Hürden in ihrem Curriculum, unter anderem, weil diese Veranstaltungen nicht nur mit einer klassischen Klausur abgeschlossen werden, sondern auch mit einer praktischen Programmierprüfung am Rechner [Sc17].

Programmieren Können ist keine rein intellektuelle Fähigkeit, sondern hat auch einen sehr hohen handwerklichen Anteil. Ähnlich wie bei humanoiden Muskeln kann durch regelmäßiges Trainieren des „Programmiermuskels“ eine höhere Leistungsfähigkeit erzielt werden. Deshalb sind bei den Programmierveranstaltungen im Department Informatik der HAW Hamburg die Praktika ein zentraler Bestandteil, in dem der aktive Umgang mit Quelltexten geübt und überprüft werden kann. Die Erfahrung zeigt aber auch, dass der Umfang des Praktikums bei weitem nicht ausreicht, um signifikant Programmiererfahrung sammeln zu können. Ein hoher Eigenbeitrag der Studierenden außerhalb der Präsenzveranstaltungen ist für einen erfolgreichen Abschluss notwendig.

Auf diversen Online-Plattformen wie *HackerRank* [HR22], *Codewars* [CW22] oder *edabit* [eda22] stehen zwar viele Programmieraufgaben zum Üben zur Verfügung; aber das Suchen und Finden geeigneter Aufgaben, die zum eigenen Lernfortschritt passen, ist angesichts

¹ HAW Hamburg, Dep. Informatik, Berliner Tor 7, 20099 Hamburg, Deutschland, axel.schmolitzky@haw-hamburg.de

² HAW Hamburg, Dep. Informatik, Berliner Tor 7, 20099 Hamburg, Deutschland, henri.burau@haw-hamburg.de

eines straff konzipierten Vollzeitstudiums von den meisten Studierenden nicht leistbar, insbesondere bei mangelnder Programmiererfahrung, wie sie in den kritischen Einstiegsveranstaltungen der ersten Semester vorherrscht. Seit 2020 entwickelt ein Projektteam des Departments Informatik an der HAW Hamburg deshalb eine Online-Plattform, auf der Studierende ergänzend zu unseren Programmierveranstaltungen mit passenden Aufgaben üben können. In diesem Artikel stellen wir die Entwurfsprinzipien und die Architektur dieser Plattform vor.

2 Grundkonzepte von Programmierübungsplattformen

Im Folgenden kürzen wir Online-Plattformen, auf denen Programmieren geübt werden kann, mit OPP ab, als Akronym für das englische *Online Programming Practice platform*. Wir betonen den Begriff *Practice*, also Üben, um einerseits eine Abgrenzung zu allgemeinen Online-Programmierplattformen zu verdeutlichen (Online-Versionen von *Visual Studio Code* oder *Eclipse* beispielsweise sind vollwertige Online-IDEs, die aber keinen Lehanspruch erheben) und andererseits als Abgrenzung zu Online-Lernsystemen, auf denen Programmierkonzepte systematisch gelehrt werden bzw. gelernt werden können (mit ergänzenden Folien, Videos, etc.). Eine OPP bietet somit in der von uns verstandenen Reinform keine Lehrinhalte an, sondern fokussiert auf möglichst effektives Üben.

Eine OPP hat folglich als ein zentrales Konzept die *Aufgabe*. Eine Person, die auf einer Plattform eine solche Aufgabe bearbeitet, reicht einen *Lösungsversuch* in Form von Programmtext ein. Eine zentrale Anforderung an eine OPP ist, dass sie *automatisiert Feedback* zu Lösungsversuchen liefert. Bei inkorrekten Lösungsversuchen geht es um geeignetes Feedback zur statischen Übersetzbarkeit und zum dynamischen Verhalten, dies ist essenzieller Bestandteil einer OPP. Aber auch bei korrekten Lösungsversuchen kann über eine OPP Feedback gegeben werden, beispielsweise zur Qualität der gelieferten Lösung: im Vergleich zu anderen über Rankings, als Bewertung der Speicher- und/oder Laufzeiteffizienz oder schlicht als Hinweis auf alternative Lösungen, die dem System bereits bekannt sind.

Aufgaben können *sprachunabhängig* sein oder *sprachspezifisch*. Die meisten Plattformen ermöglichen sprachunabhängige Aufgaben, indem sie den Input für einen Lösungsversuch über Standard-Input liefern und den (zu bewertenden) Output auf Std-Out erwarten. Diese beiden Kanäle existieren in jeder Programmiersprache. Folglich kann beispielsweise auf der Plattform *CodingGame* [CG22] bei vielen Aufgaben aus einer Vielzahl unterstützter Programmiersprachen für den Lösungsversuch gewählt werden. Ein Nachteil solcher Aufgaben ist, dass ein teilweise beträchtlicher Anteil in das Parsing der Eingabe und die korrekte Formatierung der Ausgabe investiert werden muss. Dieser Aufwand kann bei einfacheren Aufgaben leicht die eigentlichen Lernziele „verschütten“, abgesehen davon, dass Programmieren über Std-In-Out keine Praxisrelevanz besitzt.

Wenn das Verständnis von sprachspezifischen Konzepten geübt werden soll, reicht dieser generische Ansatz auf jeden Fall nicht mehr aus. Wenn beispielsweise mit einer Aufgabe

das Verständnis der Objektstruktur zweidimensionaler Arrays in Java überprüft werden soll („Ergänze in einem gegebenen zweidimensionalen int-Array von gleichlangen Zeilen eine weitere Zeile für die Summen der Spalten, erhalte aber in der Ergebnisstruktur die gelieferten Zeilen-Arrays“), dann würde ein „Plattklopfen“ der Array-Objekte in eine zweidimensionale Textstruktur dem Lernziel nicht dienen bzw. ein Überprüfen des Erfolges unmöglich machen.

3 Das Design und der Einsatz von OPPSEE

Im Sommer 2019 hat das Department Informatik der HAW Hamburg angesichts damals noch vorhandener HSP-Mittel beschlossen, eine eigene OPP für Programmieraufgaben zu entwickeln. Den Entwurfsraum für solche Plattformen haben wir 2020 auf der SEUH in Innsbruck dargestellt [GHS20] und anschließend den Namen unserer Plattform angelehnt an den Titel der Publikation als das Akronym OPPSEE für *Online Programming Practice for Software Engineering Education* gewählt.

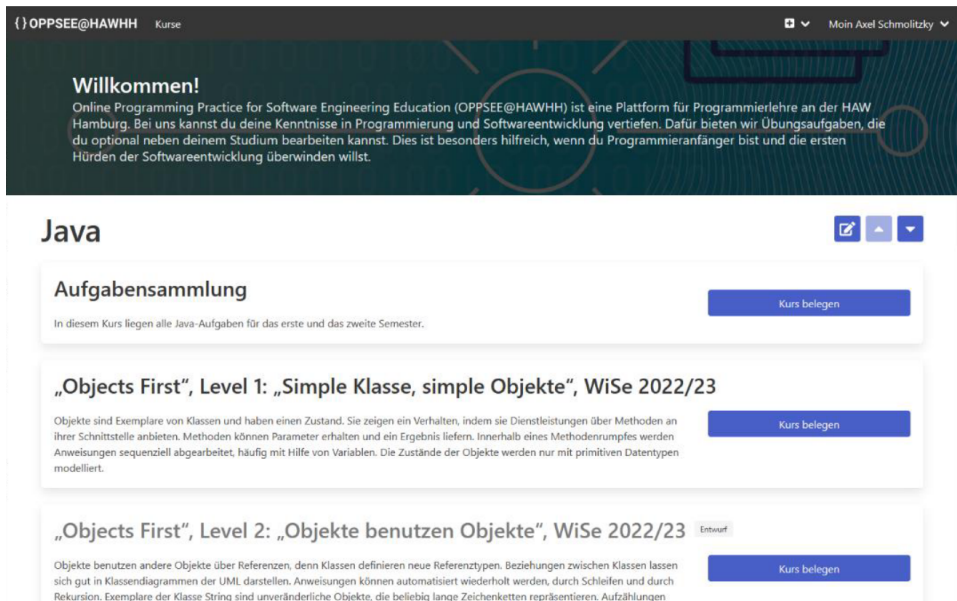


Abb. 1: Die Einstiegsseite von OPPSEE

3.1 Design

Eine zentrale Entwurfsvorgabe für OPPSEE war, dass die Plattform lediglich eine *freiwillige Ergänzung* zu bestehenden Lehrveranstaltungen darstellen, aber keine Lehre auf der

Plattform *betrieben* werden sollte. Damit ist gemeint, dass beispielsweise das Bereitstellen von Praktikumsaufgaben und die Kontrolle ihrer termingerechten Bearbeitungen explizit nicht auf der Plattform abgebildet werden sollen. Lehrveranstaltungen sollten nicht von OPPSEE abhängig sein. Auch summative Prüfungen sind auf OPPSEE derzeit bewusst nicht vorgesehen. Dies befreit den Entwurf vom Ballast aller Konzepte, die für den Nachweis der Identität und der individuellen Leistung oder das Aufspüren von Plagiaten notwendig sind. Ein weiterer Unterschied zu klassischen Lernmanagementsystemen (LMS) ist, dass wir eine weitgehende Anonymität der Benutzer anstreben. Die meisten LMS entlasten vor allem die Lehrenden, indem sie diesen Mechanismen zur Kontrolle ihrer Lernenden anbieten. Die Einstiegshürde zur Nutzung einer OPP sollte aber möglichst niedrig sein, u.a. indem ein Raum zum Üben angeboten wird, den die Studierenden angstfrei nutzen können.

Eine damit verbundene Design-Entscheidung, die den Gesamtentwurf ebenfalls erleichtert hat, war das Verwenden bestehender Authentisierungskomponenten, in diesem Fall der hochschulweiten Gitlab-Installation. Jeder Angehörige der HAW Hamburg hat eine sogenannte a-Kennung, die den Zugang zu allen digitalen Diensten ermöglicht, auch zu der Gitlab-Instanz. Benutzer können auf diese Weise sehr leichtgewichtig in OPPSEE modelliert werden und bleiben unter ihrer a-Kennungen weitgehend anonym. Dies bewirkt allerdings auch, dass die Plattform derzeit nicht ohne weiteres für die Web-Öffentlichkeit zur Verfügung gestellt werden kann.

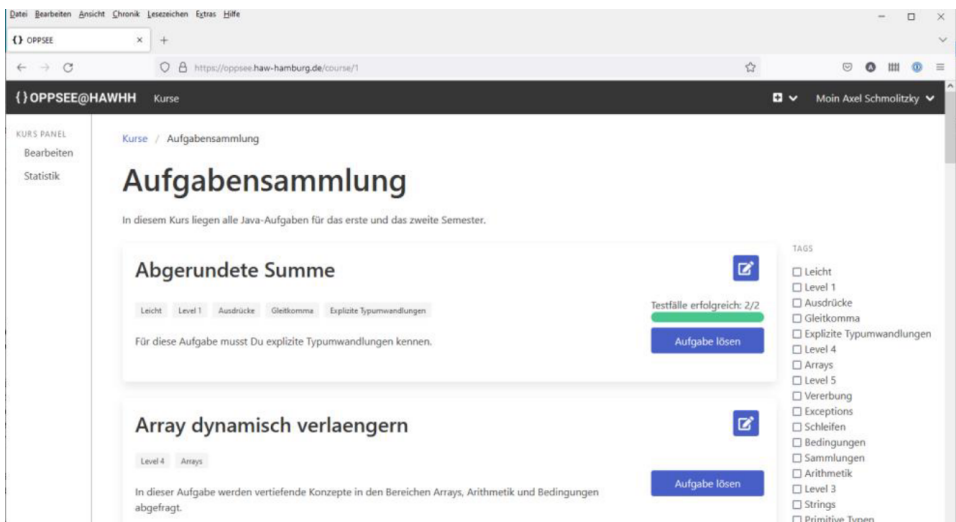


Abb. 2: Ein Kurs in OPPSEE

Eine Kernidee der Plattform ist, dass Lehrende gezielt passende Aufgaben zu ihren jeweiligen Veranstaltungen anbieten können. Sie sollen damit die Studierenden ihrer Veranstaltungen davon entlasten, sich im Web auf den diversen existierenden OPPs selbst Aufgaben suchen zu müssen, die zu ihrem aktuellen Lernfortschritt passen. Die Bündelung mehrerer Aufgaben zu diesem Zweck nennen wir auf der OPPSEE-Plattform einen Kurs. Der Begriff ist dabei leicht

irreführend, weil weder eine Anmeldung notwendig ist noch eine Zugangsbeschränkung vorgenommen wird. Jede auf OPPSEE angemeldete Person kann einen der angebotenen Kurse „belegen“ (Abb. 1) und direkt mit der Bearbeitung der verknüpften Aufgaben beginnen (Abb. 2). Die Lehrenden können die von ihnen ausgewählten Aufgaben zusätzlich mit Tags versehen, die aus Sicht des jeweiligen Kurses sinnvoll und passend sind.

Die Bearbeitung einer Aufgabe erfolgt in einer vollwertigen Online-IDE, die neben einem Editor mit Tabs, Syntax-Highlighting und Code-Completion auch einen Debugger, einen Datei-Explorer und die Anbindung an ein Git-Repository ermöglicht (Abb. 3). Der Leistungsumfang ist dabei teilweise von der unterstützten Programmiersprache abhängig. Derzeit gibt es Unterstützung für Java, C und Python. Diese Online-IDE ist keine Eigenentwicklung, es wird eine frei verfügbare IDE eingebettet (siehe nächster Abschnitt zur Architektur).

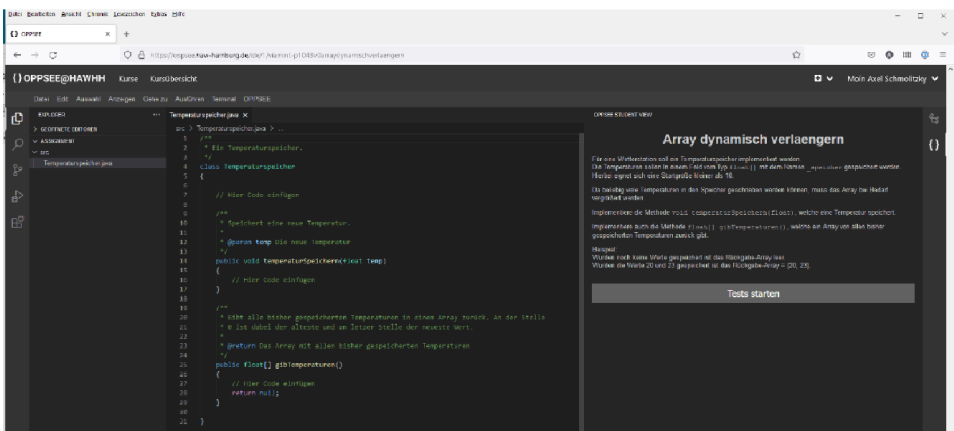


Abb. 3: Die Online-IDE in OPPSEE

Nach der erfolgreichen Bearbeitung einer Aufgabe fragt die Plattform explizit Feedback zum empfundenen Schwierigkeitsgrad, zur Verständlichkeit der Aufgabenstellung, zum Zeitaufwand und zum persönlichen Lerneffekt durch die Aufgabe ab. Die dabei erhobenen Daten werden für die Lehrenden zu Statistiken verdichtet, um die Wirksamkeit der Aufgaben besser einschätzen zu können (Abb. 4). Unter anderem setzen wir hier *Progress Networks* [Mc21] ein, mit denen Bearbeitungsversuche über die Testfälle visualisiert werden können. Diese erfordern zwar, dass es eine sinnvolle lineare Ordnung der Testfälle gibt (beispielsweise von leicht zu anspruchsvoll), liefern dann aber eine aussagekräftige Visualisierung, in der besonders häufig auftretende Abfolgen hervorgehoben sind. Auch hier liegt der Fokus auf einer Kontrolle der Aufgaben, nicht der Studierenden. Es soll keine *Leistungskontrolle* der Studierenden durch die Lehrenden implementiert werden, sondern primär für die Studierenden eine *Leistungsunterstützung*, deren Wirksamkeit für die Lehrenden ausgewertet wird.



Abb. 4: Bearbeitungsstatistik zu einer Aufgabe (Auszug)

3.2 Bisherige Einsätze

Die Plattform wurde bisher in drei Veranstaltungen eingesetzt, in denen Java vermittelt wird. Im Department Informatik waren dies eine Erst- und eine konsekutive Zweitsemesterveranstaltung zur Programmierung, im Department Informations- und Elektrotechnik eine Drittsemesterveranstaltung (dort wird in den ersten beiden Semestern C und C++ vermittelt). Ein weiterer Einsatz erfolgte in einer einführenden Veranstaltung zur Programmierung mit C im Department Wirtschaftsingenieurwesen.

Die in Abb. 1 sichtbaren Beispiele sind Kurse, die ergänzend zu einer Lehrveranstaltung nach dem Objects First-Ansatz [BK16] angeboten wurden, deren Inhalte im Sinne einer Gamification in vier Level eingeteilt sind und deren didaktische Grundlagen bereits auf der SEUH vorgestellt wurden [SZ07]. Auf jedem Level werden Programmierkonzepte eingeführt, die in den ergänzenden OPPSEE-Kursen gezielt geübt werden können. Eine Idee dieser expliziten Level ist, dass Sprachkonzepte, die auf einem späteren Level eingeführt werden, nicht für die Lösung von Aufgaben auf früheren Levels eingesetzt werden dürfen. Beispielsweise werden Arrays als Konzept erst auf Level 4 eingeführt und sind deshalb nicht zugelassen als Lösungsbestandteil von Aufgaben auf Level 1 bis 3. Dieser grundsätzliche didaktische Ansatz, der in der Lehrveranstaltung schon seit Jahren in Vorlesung und Praktikum gelebt wird, wird derzeit noch nicht in den OPPSEE-Aufgaben technisch erzwungen. Die Plattform ermöglicht es zwar über eine mögliche Quelltextanalyse, dies ist aber bisher nicht umgesetzt.

Dies liegt unter anderem daran, dass der Aufwand für die *Erstellung* von Aufgaben für die Lehrenden vergleichsweise hoch ist. Erst wenn ein Fundus an Aufgaben existiert, wird es möglich sein, dass Lehrende nur noch passende Aufgaben für ihre Veranstaltung zusammenstellen müssen. Dieses schlichte *Zusammenstellen* von bestehenden Aufgaben

zu einem Kurs ist innerhalb der Plattform sehr komfortabel umgesetzt. Am Beispiel der Objects First-Veranstaltung wird auch deutlich, dass es zu einer Lehrveranstaltung durchaus mehrere “Kurse” geben kann. Eine weitere didaktische Idee der Einteilung in Level ist deshalb, dass Studierende am Ende eines Levels Feedback über ihren bisherigen Lernerfolg erhalten können, indem sie die Aufgaben zu diesem Level zum vertiefenden Üben bearbeiten. Idealerweise geschieht dies semesterbegleitend; allerdings zeigt die Erfahrung, dass viele Studierende doch erst kurz vor der Prüfung von dieser Möglichkeit Gebrauch machen.

4 Die Architektur von OPPSEE

Die Plattform wurde als eine Micro-Service-Architektur entworfen, um eine leicht erweiterbare und zuverlässige Anwendung zu erhalten. Dabei werden sowohl die Microservices als auch die Online-IDEs in einem Kubernetes-Cluster betrieben. Abb. 5 zeigt das Zusammenspiel der Microservices in einem Komponentendiagramm. Drei Komponenten können als Subsysteme identifiziert werden: Das Dashboard, die Online IDE und das OPPSEE Backend. Nutzer interagieren mit der Plattform über das Dashboard oder die Theia-IDE. Dabei dient das Dashboard zur Verwaltung und Auswahl der Kurse und Aufgaben, die auf OPPSEE zur Verfügung stehen. Die Bearbeitung der Aufgaben geschieht in der Online IDE. Das OPPSEE-Backend ist für das Speichern der Daten sowie für die Bereitstellung der IDEs verantwortlich. In den folgenden Abschnitten werden die Komponenten näher erläutert.

4.1 Dashboard

Als primärer Einstiegspunkt in OPPSEE dient das Dashboard. Nach der Anmeldung können Studierende dort die Aufgaben zur Bearbeitung auswählen. Die Auswahl wird dabei durch die Organisation der Aufgaben in Kategorien und Kurse, sowie durch verschiedene Filtermöglichkeiten erleichtert. Die Aufgabenerstellenden haben ihrerseits die Möglichkeit neue Aufgaben zu erstellen und diese in Kurse und Kategorien einzuteilen. Anhand der Bearbeitung und dem Feedback der Studierenden werden außerdem Statistiken erstellt und dargestellt, die Aufschluss über die Qualität der Aufgaben geben.

4.2 Online IDE

In der Online IDE erfolgt die Aufgabenbearbeitung der Studierenden. In Zukunft soll dort außerdem die Erstellung der Aufgaben möglich sein. Als Grundlage dient dabei Theia, eine IDE inspiriert von Visual Studio Code, welche komplett im Browser operiert. Klassische Funktionen einer IDE wie das Verwalten von Quellcode, Syntaxhighlighting und das Nutzen des „Language Server Protocols“ für Autovervollständigung sind bereits in Theia vorhanden. Ergänzend wurde eine Anbindung an das OPPSEE-Backend hinzugefügt.

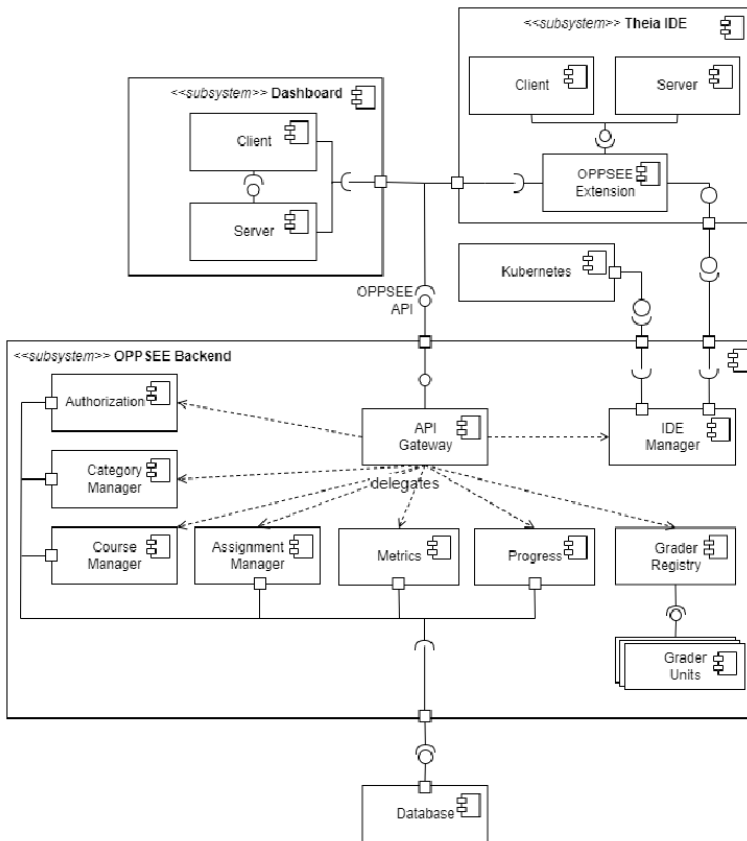


Abb. 5: Überblick der Microservice-Architektur von OPPSEE

Dadurch können Aufgaben direkt in der IDE angezeigt und zur Bewertung freigegeben werden. Auch das Feedback einer ausgewerteten Aufgabe wird direkt in der IDE dargestellt. Da die Schnittstelle für die Online IDE auch von lokalen IDEs wie Visual Studio Code oder Eclipse implementiert werden kann, besteht die Möglichkeit OPPSEE-Aufgaben auch lokal zu bearbeiten.

4.3 OPPSEE Backend

Das OPPSEE Backend stellt eine REST-Schnittstelle zur Verfügung, worüber Online-IDE und Dashboard Zugriff auf die unterschiedlichen Services erhalten. Als zentraler Einstiegspunkt dient dabei das skalierbare „API-Gateway“, welches die einzelnen Anfragen auf

korrekte Autorisierung überprüft. Die einzelnen Services registrieren dort ihre Dienstleistungen dynamisch, um eine leicht erweiterbare Architektur zu schaffen. Eine zentrale Aufgabe übernimmt der „IDE-Manager“, der die Online-IDEs verwaltet. Für jede Aufgabe und Studierenden wird eine eigene IDE gestartet, die in Form eines Kubernetes-Pods betrieben wird. Bearbeiten also 60 Studierende auf der Plattform Aufgaben, werden 60 Pods gestartet. Nach einer festgelegten Dauer von Inaktivität wird der Quelltext gesichert und der Pod heruntergefahren. Um Ladezeiten zu reduzieren und die vorhandene Infrastruktur nicht zu überlasten, wird es außerdem vermieden zwei Pods mit dem gleichen Image zu starten, stattdessen können die Aufgaben in einem bereits gestarteten Pod ausgetauscht werden. Das Erzeugen von automatischem Feedback wird durch die Grader Registry sichergestellt. Dort können sich Grader Units registrieren, die ihrerseits Feedback zu Aufgaben erstellen. Grader Units melden dabei ihren Namen sowie die unterstützten Programmiersprachen. Dieses flexible System einer Grader-Pipeline erlaubt es, eine Aufgabe von mehreren, unterschiedlichen Bewertungssystemen auswerten zu lassen, um verschiedene Aspekte des Quellcodes zu bewerten. Neben funktionalen Unit-Tests und Bewertungen der statischen Quelltext-Qualität sind beispielsweise auch Bewertungen der Laufzeit- oder Speichereffizienz wünschenswert, um Aspekte wie Nachhaltigkeit beim Programmieren zu adressieren.

5 Verwandte Systeme

Es gibt einige Systeme aus dem Hochschulkontext, die ähnliche Funktionalität zur Verfügung stellen. In diesem Abschnitt werden relevante Systeme vorgestellt.

5.1 Artemis

ArTEMiS ist ein in Java geschriebenes und an der TU München entwickeltes Open-Source „Automated assessment Management System“ für interaktives Lernen. Es kombiniert Versionskontrolle und kontinuierliche Integration mit automatischer Bewertung von Programmierübungen und sofortigem Feedback [KS18]. Die Studierenden können ihre Lösungen sowohl über den Online-Editor als auch über eine lokale IDE einreichen. Die Architektur von ArTEMiS ist sehr flexibel und sprachunabhängig: Sie erlaubt jede Programmiersprache, die in eine CI-Pipeline eingebaut werden kann. ArTEMiS bietet außerdem fortgeschrittene Funktionen wie die Analyse von UML-Diagrammen. Es unterstützt auch die Verwaltung von Aufgaben für Teams. Es hat sich als robustes und flexibles System erwiesen, das in großen Kursen eingesetzt werden kann. Die Zielrichtung von Artemis ist aber eine andere als bei OPPSEE. Artemis ist explizit für das Betreiben von Lehrveranstaltungen konzipiert, indem Studierende und formativ bewertete Aufgaben samt den Deadlines ihrer Abgabe modelliert sind. Das System bietet Studierenden keine voll funktionsfähige IDE ohne eine lokale Installation und die Kenntnis von Git. Wenn ArTEMiS für freiwillige Zusatzaufgaben verwendet werden würde, wären diese Hürden für Studierende möglicherweise zu hoch, insbesondere für Programmieranfänger.

5.2 JACK

JACK ist ein Framework für modulare Benotung und Feedbackgenerierung, das an der Universität Duisburg-Essen entwickelt wurde. Der Fokus von JACK liegt auf nützlichem Feedback. Lösungsversuche können direkt in JACK hochgeladen oder über ein Plugin für Eclipse eingereicht werden. Zum Testen des eingereichten Codes wird eine breite Palette von “Checkern” angeboten, die den Code mit verschiedenen Metriken analysieren können. Dazu gehören sowohl dynamische Tests als auch statische Code-Analysen [St16]. JACK kann ohne Git-Kenntnisse verwendet werden, erfordert aber eine lokale IDE-Installation. Das Einrichten und Anpassen eines Kurses in JACK ist eine recht komplexe Aufgabe, selbst mit einer großen Anzahl von bereits verfügbaren Aufgaben, so dass hier eher die Hürde für Lehrende recht hoch ist.

5.3 Code Freak

Code FREAK (Code Feedback, Review & Evaluation Kit) [CF22] ist eine Open Source Plattform zur Bearbeitung und Feedbackgenerierung, die an der Fachhochschule Kiel genutzt wird. Code FREAK ist agnostisch gegenüber Programmiersprachen und bietet eine Online-IDE, in der die Aufgaben bearbeitet werden können. Die Aufgaben können alternativ auch durch das Hochladen von Lösungsversuchen bearbeitet werden. Durch eine Implementation des LTI Standards ist eine Anbindung an moderne „Learn Management Systems“ (LMS) wie Moodle möglich.

5.4 Codeboard

Codeboard [CB22] wurde an der ETH Zürich entwickelt und dort eingesetzt. Es erlaubt das Erstellen von Online-IDEs mit vorgegebenen Dateien. Der geschriebene Quelltext kann dann automatisch ausgewertet werden. Die Online-IDE ist dabei für alle Programmiersprachen nutzbar, verzichtet aber auf die Integration des „Language Server Protocols“ und damit auf viele Werkzeuge moderner IDEs, wie beispielsweise eine Autovervollständigung. Damit die Ergebnisse der Tests auch in LMS aufgenommen werden können, unterstützt Codeboard den LTI Standard. Eine Unterstützung für die Darstellung von Aufgabentexten ist nicht vorhanden.

6 Die weiteren Ziele von OPPSEE

Angesichts der seit dem Startschuss 2019 veränderten Finanzsituation an der HAW Hamburg mussten wir als Entwicklungsteam zwischenzeitlich den Schwerpunkt auf kleine Aufgaben legen, die primär für einführende Veranstaltungen geeignet sind. Auf diese Weise konnte ein

hochschulweiter Nutzen der Plattform propagiert werden, weil es in etlichen Studiengängen außerhalb des Departments Informatik Veranstaltungen zu Programmiergrundlagen gibt, die einen Einstieg mit C, Python oder Java machen und bei denen OPPSEE ergänzend hilfreich sein kann. Hier sollen die bereits bestehenden Kontakte weiter ausgebaut werden.

Angetreten waren wir jedoch mit dem Anspruch eine OPP zu entwerfen, auf der auch fortgeschrittene Konzepte der Softwareentwicklung geübt werden können: Programmieren in großen Systemen, Entwickeln im Team, Verwenden eines Repositories. Eine zentrale Herausforderung wird somit sein, Aufgaben so zu entwerfen, dass sie von ihrem Umfang innerhalb einer vorgegebenen Zeit nur im Team erfolgreich bearbeitet werden können, indem eine geeignete Modularisierung und Verteilung vorgenommen wird. Dazu werden etliche Anpassungen an der Plattform notwendig werden.

Auf etwas abstrakterer Ebene ist die Entwicklung einer “Theorie” für Programmieraufgaben interessant: Wie kann der Austausch von Aufgaben unter Lehrenden erleichtert werden? Wie kann dieselbe Aufgabe in verschiedenen didaktischen Ansätzen unterschiedlich ausgeprägt sein? Wie können Varianten einer Aufgabe geeignet modelliert werden, um Redundanzen zu vermeiden?

Sobald ausreichend Nutzungsdaten vorliegen, werden Rankings wie beispielsweise auf CodingGame für die Motivation der Benutzer interessant. Dies ist eine der Wunschvorgaben aus dem Department, die wir bisher nicht umsetzen konnten.

Eine weitere Zielrichtung ist die Entwicklung von visuellen Gradern, um ähnlich ansprechende Visualisierungen von Lösungsversuchen anzubieten wie auf CodinGame. Die Grader-Architektur ist bereits auf solche Möglichkeiten ausgerichtet.

7 Zusammenfassung

OPPSEE ist eine neue Online-Plattform zum Programmieren Üben. Sie legt explizit einen Fokus auf Aufgaben und ihre Bewertung, die als Ergänzung zu Lehrveranstaltungen eingesetzt werden können. OPPSEE soll bestehende Lehre unterstützen, aber es soll keine Lehre auf OPPSEE betrieben werden. Für Lehrende wird das Bündeln von bestehenden Aufgaben in für die eigene Veranstaltung in passende Einheiten sehr einfach gemacht. Die Architektur der Plattform kombiniert eine vollwertige Online-IDE mit einem Backend zur Aufgabenverwaltung und zur Überprüfung von Lösungsversuchen. Durch die Grader-Pipeline können sehr flexibel verschiedene Aspekte eines Lösungsversuchs überprüft werden.

Literatur

- [Be21] Bender, E.; Barbas, H.; Hamann, F.; Soll, M.; Sitzmann, D.: Fähigkeiten und Kenntnisse bei Studienanfänger*innen in der Informatik: Was erwarten die

- Dozent*innen? Ergebnisse einer deutschlandweiten Umfrage unter Informatik-Hochschuldozent*innen. In: 9. Fachtagung Hochschuldidaktik Informatik (HDI), Dortmund. Sep. 2021.
- [BK16] Barnes, D. J.; Kölling, M.: Objects First with Java: A Practical Introduction Using BlueJ. Prentice Hall Press, USA, 2016, ISBN: 0132492660.
- [CB22] Codeboard, 2022, URL: <https://codeboard.io>, Stand: 06. 11. 2022.
- [CF22] Code FREAK, 2022, URL: <https://codefreak.org>, Stand: 06. 11. 2022.
- [CG22] CodinGame, 2022, URL: <https://www.codingame.com>, Stand: 06. 11. 2022.
- [CW22] Codewars, 2022, URL: <https://www.codewars.com>, Stand: 06. 11. 2022.
- [eda22] edabit, 2022, URL: <https://edabit.com>, Stand: 06. 11. 2022.
- [GHS20] Gandraß, N.; Hinrichs, T.; Schmolitzky, A.: Towards an Online Programming Platform Complementing Software Engineering Education. In (Krusche, S.; Wagner, S., Hrsg.): Tagungsband des 17. Workshops “Software Engineering im Unterricht der Hochschulen” 2020, Innsbruck. Bd. 2531. CEUR Workshop Proceedings, S. 27–35, Feb. 2020.
- [HR22] HackerRank, 2022, URL: <https://www.hackerrank.com>, Stand: 06. 11. 2022.
- [KS18] Krusche, S.; Seitz, A.: ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education. SIGCSE ’18, Association for Computing Machinery, Baltimore, Maryland, USA, S. 284–289, 2018, ISBN: 9781450351034, URL: <https://doi.org/10.1145/3159450.3159602>.
- [Mc21] McBroom, J.; Paassen, B.; Jeffries, B.; Koprinska, I.; Yacef, K.: Progress Networks as a Tool for Analysing Student Programming Difficulties. In: Australasian Computing Education Conference. ACE ’21, Association for Computing Machinery, Virtual, SA, Australia, S. 158–167, 2021, ISBN: 9781450389761, URL: <https://doi.org/10.1145/3441636.3442366>.
- [Sc17] Schmolitzky, A.: Zahlen, Beobachtungen und Fragen zur Programmierlehre. In (Bruegge, B.; Krusche, S., Hrsg.): Tagungsband des 15. Workshops “Software Engineering im Unterricht der Hochschulen” 2017, Hannover, 22. - 23. Februar 2017. Bd. 1790. CEUR Workshop Proceedings, CEUR-WS.org, S. 83–90, 2017, URL: <http://ceur-ws.org/Vol-1790/paper10.pdf>.
- [St16] Striwe, M.: An architecture for modular grading and feedback generation for complex exercises. Science of Computer Programming 129/, Special issue on eLearning Software Architectures, S. 35–47, 2016, ISSN: 0167-6423.
- [SZ07] Schmolitzky, A.; Züllighoven, H.: Einführung in die Softwareentwicklung - Softwaretechnik trotz Objektorientierung? In: Software Engineering im Unterricht der Hochschulen 10, Stuttgart, dpunkt-Verlag. S. 87–100, 2007.

An Intelligent Tutoring System Concept for a Gamified e-Learning Platform for Higher Computer Science Education

Niklas Meißner¹, Sandro Speth², Uwe Breitenbücher³

Abstract: Intelligent Tutoring Systems (ITSs) are increasingly used in modern education to automatically give students individual feedback on their performance. The advantage for students is fast individual feedback on their answers to asked questions, while lecturers benefit from considerable time savings and easy delivery of educational material. Of course, it is important that the provided feedback is as effective as direct feedback from the lecturer. However, in digital teaching, lecturers cannot assess the student's knowledge precisely but can only provide information on which questions were answered correctly and incorrectly. Therefore, this paper presents a concept for integrating ITS elements into the gamified e-learning platform *IT-REX* so that the feedback quality can be improved to support students in the best possible way.

Keywords: e-Learning; Intelligent Tutoring Systems; Feedback Strategies; Spaced Repetition; IT-REX

1 Introduction, Motivation, and Research Questions

In today's education system, more and more content is being taught to students via digital media. This has also been driven by the COVID-19 pandemic, which has primarily shifted classroom teaching to the computer or smartphone at home. However, universities are still trying to adapt to digital education efficiently. Although it may seem essential to increase students' morale, the resulting learning effect must be significant enough to be worthwhile. Therefore, to improve the effectiveness of digital tasks and assignments, it is necessary to evaluate all students' results and help them progress by guiding them in the right direction. To accomplish this, students must be offered individualized feedback and guidance based on their level of knowledge so they can further improve [HT07]. The interaction between lecturer and student in digital education leads to two major issues: (1) Improve lecturer feedback since it takes more time and is more complex in digital education than in face-to-face classes. If feedback is not given optimally, it tends to be less accurate and does not optimally support students [Su20]. (2) Increase student motivation to engage with the educational material and catch up on any deficits. Thereby, students often lack transparency in areas of teaching they have deficits and in areas they are already doing well [CLG10]. The objective is to achieve student motivation and participation with gamification elements from the IT-REX platform [Sp22].

¹ University of Stuttgart, Institute of Software Engineering, Germany niklas.meissner@iste.uni-stuttgart.de

² University of Stuttgart, Institute of Software Engineering, Germany sandro.speth@iste.uni-stuttgart.de

³ Reutlingen University, Germany uwe.breitenbuecher@reutlingen-university.de

E-learning platforms are used to digitally distribute educational material to students and provide the opportunity to, e.g., upload and collect assignments, solve tasks, and view grades. The purpose of an *Intelligent Tutoring System (ITS)* is to supplement human lecturers with intelligent systems so that feedback to students can be expressed in an individualized and precise manner [WHK20]. ITSs are used not only to support students with individualized feedback but also to assess student performance and indicate the next steps accordingly.

This paper considers the gamified e-learning platform *IT-REX* [Sp22] as a basis for further integration of the ITS concept. Thereby, we explore how ITS elements could be used to improve digital teaching. This leads to our research question for this paper:

RQ1: *"How could an Intelligent Tutoring System help to improve the quality of feedback to students compared to established e-learning platforms?"*

2 Integration of the ITS Concept into the Gamified E-Learning Platform IT-REX

As described in the previous section, e-learning platforms and the ITS concept complement each other but are not systematically integrated into a single platform. Therefore, we provide a concept of how both approaches can be combined: We propose as a basis *IT-REX* [Sp22], a gamified e-learning platform that we have designed in previous work. Now we want to enhance *IT-REX* with an ITS concept. *IT-REX* enables lecturers to provide *lecture slides*, *videos*, and *quizzes* in an easy and structured way. However, an ITS needs more in-depth information about the relationships between these educational materials. Existing ITS in the field of computer science are limited to the complementation of textbooks, in which the fixed and consistent content is carried out without a teacher having access to it or being able to adjust it.

To capture relationships between educational material, we propose using tags to link chapters in lecture slides to videos, quizzes, and other material. Tags can also be used to create hierarchies between lecture topics, which can be used to navigate the student back to the introductory chapters if they get stuck on a particular topic. Furthermore, tags allow the ITS to provide detailed feedback to recommend the fitting material for the students to close their knowledge gaps. In addition, we propose that all educational materials include a textual description of the competencies in a lecture video or script to give students a quick overview of what they have learned. This initially requires much input from lecturers but can be addressed later, e.g., with artificial intelligence (AI).

Abb. 1 presents an overview of our ITS concept. In the remainder, whenever we write „ITS“, we mean the combined ITS components proposed by this work and included in *IT-REX*. This includes the student's progress through a course, how the student improves skill levels within a course, learns new material with learning strategies, and consolidates existing

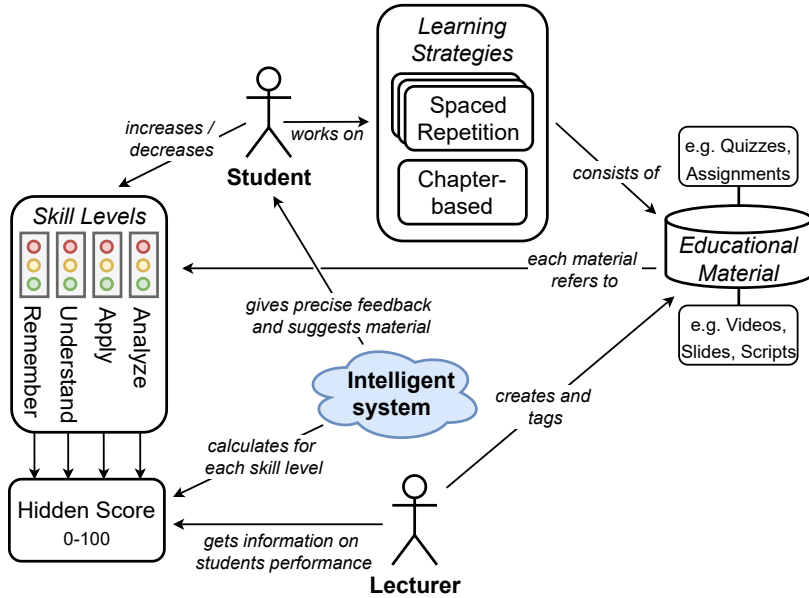


Abb. 1: Interaction of ITS components in the e-learning platform.

knowledge with repetitive tasks. The concept also shows the criteria used to measure and assess the student and how this affects the system's feedback to the student.

2.1 Skill Levels

Depending on learning taxonomies, the ITS assesses the students' skills, giving them an overview of the areas in which they need to improve. In learning theory, the learning objectives of learners are assigned to different taxonomy levels, depending on their degree of difficulty. A taxonomy is a kind of classification system. The best-known taxonomies are the „*Bloom's taxonomy levels*“ [B173]. The taxonomy levels, according to Bloom, which are focused on in this paper exclusively, include [B173]: **(Remember)** Students are able to remember what they have learned, e.g., by remembering facts, terms, or concepts. **(Understand)** Students are able to understand what they have learned, e.g., by organizing, summarizing, or describing facts. **(Apply)** Students are able to apply what they have learned, e.g., by solving problems and identifying connections in new situations. **(Analyze)** Students are able to analyze what they have learned, e.g., by examining and breaking information into components or making inferences. These learning taxonomy levels are named „*skill levels*“ in this paper and indicate different strengths and weaknesses of a student.

In our approach, we compute these learning taxonomies based on students' answers to course materials such as quizzes and assignments. Each quiz, assignment, and exercise of a

course contributes to one of the skill levels. The ITS automatically determines the progress of the corresponding skill level based on the set tags and content of the assignment. For example, suppose the ITS detects that a quiz has been answered incorrectly. In that case, it will suggest to the student educational material from previous chapters or areas that address the task's basic information, thus repeating the skill level of remembering and understanding. This association is achieved through the previously described tags and is then applied by the system. This means that if a student has already missed the precondition of a question and fails it in the task, the ITS sends the student back to the fundamentals to learn them again. This shows the student what knowledge is required to solve the corresponding task.

2.2 Learning Strategies

Our approach includes two learning strategies that teach the student the course content and provide the knowledge within the e-learning course. Depending on the different learning strategies, feedback needs to be targeted differently to students. (**Chapter-Based Learning**) In higher education, chapter-based learning is most common, where the lecturers provide educational material for each chapter, and the students work through them one after the other. The individual chapters can either build on each other in content or be completely independent but are unlocked in a specific order. After the content of a chapter (e.g., videos or scripts) has been worked through by students, they can directly check what they have learned by answering a chapter quiz. Successful quiz completion is also a prerequisite for unlocking the next chapter. (**Spaced Repetition**) Spaced Repetition uses the concept of flashcards and is designed to reinforce what has been learned previously. The students have to answer questions from the previously learned chapters in random order. In contrast to the „*Leitner System*“ [Le01], commonly used for flashcards, this concept does not manage the questions in virtual boxes. Instead, there is a question pool containing all the questions from the chapters worked on. If a question is answered correctly, it is added to the question pool after a specific time. If, on the other hand, a question was answered incorrectly, it is directly added back to the question pool.

2.3 Scoring System

Since students should not know the assessment and scoring system in the background of the ITS, it is called „*hidden score*“. Note that individual student performance may vary by chapter. This enables chapter-based feedback to be provided to each student. The hidden scores are, therefore, not only comprehensive for an entire course but also for the individual learning taxonomy levels. Thus, the student's performance at the individual levels is calculated and can be displayed individually per chapter within a course. Its value is in the range between 0 and 100. In calculating the score, several pieces of information must be considered to calculate an appropriate increase or decrease of the score. For this, we have developed the following metrics: (1) The hidden score grows less quickly if the student uses

hints when answering a question. (2) The more frequently the same question is answered in a short period, the slower the hidden score grows. (3) If a question is answered correctly many times and then incorrectly once, the hidden score drops slowly at first. In addition, other metrics can be introduced into the score calculation, e.g., how much time was needed for a task or how often the answer was changed and corrected.

To show students their progress, traffic lights are used, which, depending on their color, describe the current state of knowledge. The color of the traffic light depends on the value of the respective hidden score. These visually appealing elements are key components of the gamified e-learning platform.

2.4 Feedback Strategy

Feedback is essential for improving knowledge and acquiring skills in educational contexts. The general purpose is to change student thinking or behavior to improve learning. There may also be feedback for lecturers regarding the effectiveness or efficiency of their instructions, but this paper focuses exclusively on feedback for students. While it is considered that well-structured feedback can drastically improve student learning, it also emphasizes that feedback must be given correctly and at the right time [FG11, Sh08]. The following types of feedback are used in our approach, ranked by complexity: **(Verification)** Tells the students whether their answer was correct or incorrect, also known as „*knowledge of outcome*“. **(Correct Response)** Tells the students the correct answer without additional information, also known as „*knowledge of correct response*“. **(Try again)** If an incorrect answer is given, the students have one or more attempts to correct it, also known as „*repeat-until-correct*“. **(Topic contingent)** Provides more detailed feedback on the general topic of the question and may take the form of a simple repetition of the material. **(Response contingent)** Provides more detailed feedback based on the student's specific answer, usually giving the student information about why their answer was wrong while explaining the correct answer. These types of feedback can be used in combination with each other and with different timings - immediate or delayed. Which feedback each student receives is decided at different layers. This is done in the form of a decision tree and includes the following layers: **(Student achievement)** The overall performance of a student. **(Task level)** The difficulty of the task; divided into a lower and a higher level. **(Timing of feedback)** The timing of the feedback, either immediate or delayed. **(Prior knowledge)** Students' prior knowledge through the previous completion of the task, divided into low and high. **(Amount of repetitions)** The number of repetitions depending on prior knowledge. **(Hint types)** The types of hints are divided into three different categories: (1) Orientation guides attempt to summarize or describe the overall topic and present the problem space in a more precise picture. (2) Instrumental hints attempt to provide more specific advice to the student based on individual questions. (3) Bottom-out hints, usually the answer itself.

3 Conclusion & Future Work

The presented concept of aligning with feedback strategies in a gamified e-Learning platform to achieve the best possible success could greatly help students increase their success in learning. While other ITS approaches target a specific domain and tailor it to it, this concept is domain-independent and could fill any potential scope. To evaluate our research question, we need to analyze student responses, assess learning outcomes in a case study, and examine student reactions to various forms of feedback. This case study is part of our future work.

At the current state of the concept, the lecturers still have to give a lot of input into the ITS to make it work. This could gradually move toward using more AI and thus being trained to automatically mark videos and quizzes to link them together. Using AI will make the ITS smarter and take additional work away from the lecturers. Another possibility would be to implement and use a filter to detect irregularities and errors and ensure error tolerance. In relation to the calculation of the hidden score by the ITS, this filter would be able to assess error tolerance so that the student's ability level can be correctly estimated even if the student should answer the same question incorrectly once after having answered it correctly several times.

Literaturverzeichnis

- [Bl73] Bloom, B.S.: Taxonomie von Lernzielen im kognitiven Bereich. Beltz, 1973.
- [CLG10] Chen, Pu-Shih Daniel; Lambert, Amber D.; Guidry, Kevin R.: Engaging online learners: The impact of Web-based learning technology on college student engagement. *Computers & Education*, 54(4):1222–1232, 2010.
- [FG11] Fernando Gutierrez, John Atkinson: Adaptive feedback selection for intelligent tutoring systems. 38(5):6146–6152, 2011.
- [HT07] Hattie, John; Timperley, Helen: The Power of Feedback. *Review of Educational Research*, 77(1):81–112, 2007.
- [Le01] Leitner, S.: So lernt man lernen: der Weg zum Erfolg. Herder-Spektrum. Herder, 2001.
- [Sh08] Shute, Valerie: Focus on Formative Feedback. *Review of Educational Research*, 78:153–189, 03 2008.
- [Sp22] Speth, Sandro; Becker, Steffen; Breitenbücher, Uwe; Fuchs, Philipp; Meißner, Niklas; Riesch, Anna; Wetzel, Daniel: IT-REX — A Vision for a Gamified e-Learning Platform for the First Semesters of Computer Science Courses. In: *Software Engineering im Unterricht der Hochschulen (SEUH 2022)*. Gesellschaft für Informatik, Bonn, S. 43–48, 2022.
- [Su20] Sufyan, Agus; Nuruddin Hidayat, Didin; Lubis, Amany; Kultsum, Ummi; Defianty, Maya; Suralaga, Fadhilah: Implementation of E-Learning During a Pandemic: Potentials and Challenges. In: *2020 8th International Conference on Cyber and IT Service Management (CITSM)*. S. 1–5, 2020.

- [WHK20] Weitekamp, Daniel; Harpstead, Erik; Koedinger, Ken: An Interaction Design for Machine Teaching to Develop AI Tutors. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. CHI '20, Association for Computing Machinery, New York, NY, USA, S. 1–11, 2020.

Autorenverzeichnis