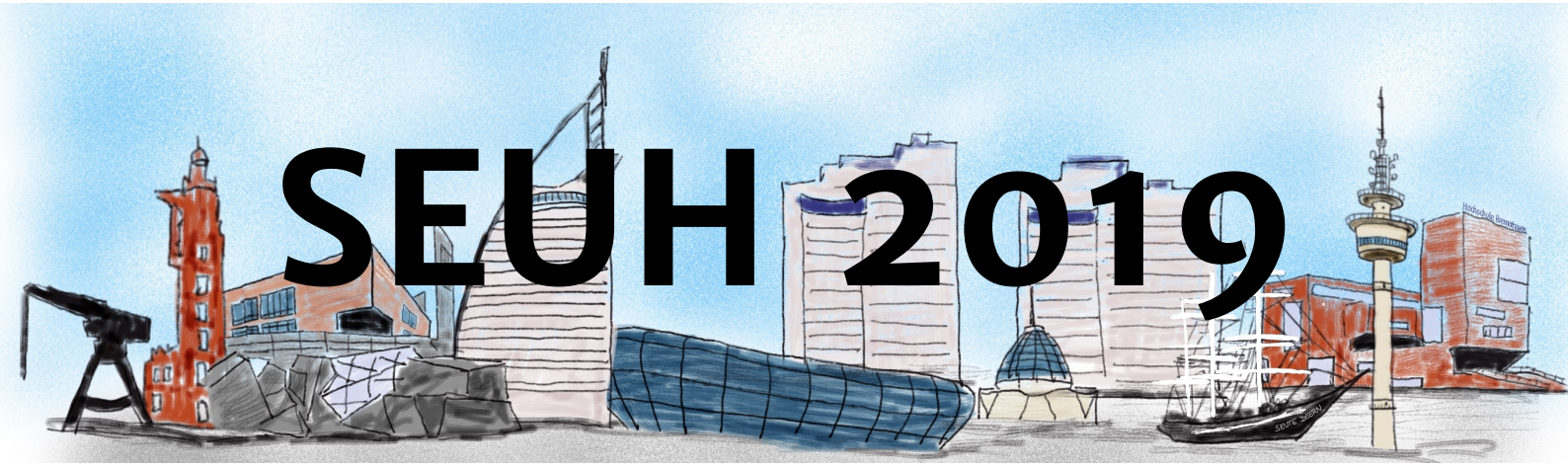
16. Workshop
Tagungsband



Software Engineering im Unterricht der Hochschulen

21. und 22. Februar 2019
Hochschule Bremerhaven

Qualität im Fokus der Software Engineering Ausbildung

Herausgeberschaft
Veronika Thurner, Hochschule München
Oliver Radfelder, Hochschule Bremerhaven
Karin Vosseberg, Hochschule Bremerhaven

Inhaltsverzeichnis

*

Vorwort

Dieser Tagungsband begleitet den 16. Workshop „Software Engineering im Unterricht der Hochschulen“ (SEUH 2019) an der Hochschule Bremerhaven.

Tradition der SEUH

Seit der ersten SEUH im Jahre 1992 haben sich einige Eigenschaften der Tagung herausgebildet, die als SEUH-typisch bezeichnet werden können und die Jochen Ludewig, einer der Gründungsväter der Tagungsreihe, im Vorwort des Tagungsbandes der SEUH 2011 sehr schön aufgelistet hat: Neben Konstanten wie einem stabilen Termin (immer Ende Februar in den ungeraden Jahren), Deutsch als Tagungssprache, einer niedrigen Teilnahmegebühr und dem informellen Vorabendtreffen ist vor allem hervorzuheben, dass dies eine Tagung zum intensiven Austausch zwischen Interessierten von Universitäten UND Fachhochschulen ist.

Thematische Schwerpunkte der SEUH 2019

Der 16. SEUH-Workshop wird gemeinsam mit der Fachtagung der GI-Fachgruppe „Test, Analyse und Verifikation von Software“ (TAV) unter dem Motto „Qualität im Fokus der Software Engineering Ausbildung “ durchgeführt.

Der Tradition folgend, schafft auch die diesjährige Tagung einen Rahmen für die Diskussion zu aktuellen Fragen der Softwaretechnik im Kontext der Hochschullehre. Diese Fragen führen zu altbekannten Themen, die wiederholt Programmpunkte seit 1992 sind, aber auch zu neuen Themen, die sich insbesondere aus Veränderungen der industriellen Softwareentwicklung und den Qualitätsanforderungen in einer sich schnell verändernden Welt ergeben. Dieses Spektrum spiegeln auch die vier eingeladenen Vorträge wider: Andreas Zeller blickt auf frische Ansätze für Forschung, Praxis und Lehre in der Testautomatisierung. Peter Zimmerer stellt vor dem Hintergrund seiner langjährigen Praxiserfahrungen die Anforderungen an Kompetenzen von Testarchitekten zur Diskussion. Während Elmar Jürgens und Eleonora Scherl Methoden des Software Engineering reflektieren, die essentielle Grundlagen für ihre agilen Projekte und in ihren Unternehmen sind.

Mit diesen Vorträgen spannen wir einen Bogen über ein breites Spektrum an Kompetenzen, die wir im Software Engineering vermitteln und stärken wollen. Mit den eingereichten Beiträgen haben wir eine Vielzahl von konkreten Beispielen, die zum Diskutieren oder auch zum Nachahmen anregen.

Das Programmkomitee wählte aus 16 Einreichungen 12 Beiträge aus, die sich überwiegend folgenden Themenschwerpunkten zuordnen lassen:

Die kritische Auseinandersetzung mit Modellen und Methoden

- Brauchen wir überhaupt noch UML in der Bachelorausbildung?
- Welche formale Methoden sollen vermittelt werden?
- Wie kann ein Werkzeug zur Unterstützung von Code-Reviewern in der Bewertung aussehen?

Kompetenzen in der Durchführung von Projekten

- Können wir Projektmanagement lernen und lehren ohne Projekte?
- Brauchen alle Projektbeteiligten algorithmisches Denken?
- Wie werden Grundfertigkeiten zur Selbstverständlichkeit?

Erfahrungen mit unterschiedlichen Lernformaten

- Wie können empirische Methoden in Projekten vermittelt werden?
- Wie können aktive Lernumgebungen gestaltet werden?
- Welche Erfahrungen gibt es mit integrierten Praxisprojekten im Studium?
- Wie definieren wir Lernziele im Software Engineering?

Agile Projekte mit crossfunktionalen Teams

- Wie werden Entscheidungen in agilen Teams verwaltet?
- Wie vermitteln wir Usability Engineering in agilen Teams?

Wir hoffen, dass die ausgewählten Beiträge auch in diesem Jahr die Grundlage für anregende Diskussionen bilden.

Dank

Eine Tagung wie die SEUH ist nicht denkbar ohne die Menschen, die die Wichtigkeit von Diskussion und Dialog zwischen Gleichgesinnten erkannt haben und dies durch ihre Unterstützung zum Ausdruck bringen. Ihnen wollen wir an dieser Stelle danken.

Zu allererst bedanken wir uns bei den Fachkolleginnen und Fachkollegen, die Beiträge eingereicht und damit eine Auswahl ermöglicht haben. Dem Programmkomitee danken wir für die Begutachtung der Beiträge, deren Einordnung in das Programm sowie das hilfreiche Feedback für die Verbesserung der Beiträge.

Karin Vosseberg und Oliver Radfelder von der Hochschule Bremerhaven und lokale Organisatoren der SEUH 2019 danken wir für die professionelle Gestaltung der Webseite, das tatkräftige Zupacken bei der Umsetzung des Tagungsbandes, die Unterstützung bei der Planung im Vorfeld sowie die hervorragende Organisation vor Ort.

Des Weiteren bedanken wir uns bei den studentischen Hilfskräften für ihre Unterstützung vor Ort bei der Durchführung der SEUH 2019.

Ebenso danken wir der Hochschulleitung der Hochschule Bremerhaven für die Möglichkeit, die Räumlichkeiten und technischen Einrichtungen der Hochschule für die SEUH zu nutzen.

Ein herzlicher Dank gebührt ferner den Sponsoren, ohne deren Unterstützung die Durchführung der SEUH in dieser From nicht möglich wäre:

- German Testing Board e.V.
- Merentis GmbH
- dpunkt Verlag
- ASQF e.V.
- GI e.V.

Den Mitarbeitern des Publikationsdienstes CEUR Workshop Proceedings danken wir für die elegante und zeitgemäße Möglichkeit, diesen Tagungsband in elektronischer Form zu veröffentlichen.

Wir freuen uns, mit diesem Tagungsband einen Einblick in die SEUH 2019 geben zu können, und wünschen anregendes und inspirierendes Lesen.

Veronika Thurner, Hochschule München
Vorsitzende des Programmkomitees

Programmkomitee der SEUH 2019

Ruth Breu, Uni Innsbruck

Bernd Brügge, TU München

Anna Hauptmann, HS Dresden

Maritta Heisel, Uni Duisburg-Essen

Marco Kuhrmann, TU Clausthal

Dieter Landes, HS Coburg

Barbara Paech, Uni Heidelberg

Oliver Radfelder (Organisation), HS Bremerhaven

Kurt Schneider, Uni Hannover

Juliane Siegeris, HTW Berlin

Veronika Thurner (Vorsitz), HS München

Karin Vosseberg (Organisation), HS Bremerhaven

Ständiges Organisationskomitee

Axel Schmolitzky, HS Hamburg

Kurt Schneider, Leibniz-Universität Hannover

Veronika Thurner, HS München

Software-Tests automatisch erzeugen – Frische Ansätze für Forschung, Praxis und Lehre

Andreas Zeller
CISPA Helmholtz-Zentrum für Informationssicherheit, Saarbrücken
zeller@cs.uni-saarland.de

Zusammenfassung

Automatisch erzeugte Softwaretests können mit wenig menschlichem Aufwand viele Fehler finden. In diesem Vortrag stelle ich aktuelle Techniken zur Testerzeugung vor, die für ein gegebenes Programm vollautomatisch dessen Eingabesprache ableiten und aus den so entstehenden Grammatiken große Mengen gültiger Testeingaben ableiten. Unser grammatikbasierter LangFuzz-Testgenerator hat so in den JavaScript-Interpretern von Firefox, Chrome und Edge tausende Fehler gefunden. Die Techniken sind in dem interaktiven Buch “Generating Software Tests” (www.fuzzingbook.org) zusammengefasst. In einer Mischung von Text und Programmcode können Leser im Browser direkt mit den Programmen experimentieren und ihren Code live ergänzen und erweitern.

Abstract

Automatically generated software tests can find many errors with little human effort. In this talk I will introduce current test generation techniques that automatically derive the input language of a given program and derive large amounts of valid test inputs from the resulting grammars. Our grammar-based LangFuzz test generator has found thousands of errors in the JavaScript interpreters of Firefox, Chrome and Edge. The techniques are summarized in the interactive book "Generating Software Tests" (www.fuzzingbook.org). In a mixture of text and program code, readers can directly experiment with the programs in their browsers and supplement and extend their code live.

Meine Lehren aus 10 Jahren Forschung und Lehre als Säulen unseres Firmenwachstums

Elmar Jürgens
CQSE GmbH, Lichtenbergstr. 8, 85748 Garching b. München
juergens@cqse.eu

Zusammenfassung

Die Firma, die ich mitgegründet habe, ist vor 10 Jahren als Spin-Off unserer Forschungsarbeiten zu Software-Qualität an der TU München entstanden. Seitdem sind wir auf 35 Mitarbeiter gewachsen, von denen die Hälfte ihre Promotion im Bereich Software-Engineering gemacht hat. Aus unseren Forschungsprototypen ist in dieser Zeit ein kommerzielles Analysewerkzeug entstanden, das täglich von Entwicklern und Testern auf der ganzen Welt eingesetzt wird.

Forschung und Lehre waren dabei nicht nur der Auslöser unserer Gründung, sondern spielen auch weiterhin eine zentrale Rolle für Innovationen und Wachstum. In dieser Keynote möchte ich meine zentralen Erfahrungen und Überraschungen aus 10 Jahren Forschung, Lehre und Firmenaufbau im Bereich Software-Qualitätsanalysen vorstellen.

Abstract

The company I co-founded 10 years ago is a spin-off of our research on software quality at the TU Munich. Since then, we have grown to 35 employees, half of whom have a PhD in software engineering. Our research prototypes have developed into a commercial analysis tool that is used daily by developers and testers all over the world.

Research and education were not only the trigger for our foundation, but also continue to play a central role in innovation and growth. In this keynote I would like to present my central experiences and surprises from 10 years of research, teaching and company building in the field of software quality analysis.

Test Architects at Siemens

Peter Zimmerer, Siemens AG, peter.zimmerer@siemens.com

Abstract

At Siemens we have invented and defined a new key role Test Architect to meet the diverse challenges of shorter time-to-market, increasing complexity and more agility while keeping quality and other key system properties high. In the real world our test systems increase in size, volume, flexibility, velocity, complexity and unpredictability: think about testing of autonomous systems or testing of AI systems (artificial intelligence). Additionally, digitalization (virtualization, cloud, mobile, big data, data analytics, internet of things, continuous delivery, DevOps) requires more than just a face lift in testing.

This talk shares our motivations, decisions, achievements, and experiences on our journey since 2016 to establish this new key role Test Architect on eye level with the software architects within the company.

Talk Description

What is a "Test Architect"? At Siemens we have invented this new key role by defining the responsibilities and tasks of a Test Architect and this talk is about our journey to establish this new key role within the company.

Imagine that you build a system with 10 million lines of code. To be successful this system should have a good architecture, and for that you need good software architects (see experience report [1]). Next, to perform good testing on 10 million lines of code, you also need to build an appropriate test system – not by spaghetti coding but with a well-designed, sustainable test architecture and by applying innovative software and test technologies.

But, who is responsible and in charge to make this happen? Typically, neither the test manager nor the quality manager will do this; therefore, we need to create a new role on eye level with the software architects: The Test Architect is born (see position paper [2]). To implement and establish this key role we have developed a unique expert training program for Test Architects at Siemens to meet the diverse challenges of shorter time-to-market, increasing complexity, and more agility while keeping quality and other key system properties high.

This talk introduces the new key role Test Architect, provides practical guidance on the needed strategies, tactics, and practices, and shares our experiences and lessons learned since the Test Architect program was launched in 2016 (see [3]):

- Why do we need a Test Architect?
- What is a Test Architect?
- What are the responsibilities and tasks of a Test Architect?
- How can a Test Architect provide value and create impact on the business?

Attend this presentation and learn what a Test Architect is all about and why we at Siemens are driving this forward. We are continuing our journey in this direction – at Siemens our target is to have nearly 50 certified Test Architects by the end of 2019. That means a tremendous upgrade and empowerment of testing!

Key Takeaways

- Understand the rationale why we need a new key role Test Architect
- Get familiar with a thorough definition of a real Test Architect – it's not just another fancy title that is misused
- Get to know the responsibilities and tasks of a Test Architect – job profile, contents, focus points
- Apply discussed strategies, tactics, practices, and experiences and become a successful Test Architect – a new career path for testers!

References

(1) Paulisch, F. and Zimmerer, P. 2010. A role-based qualification and certification program for software architects: An experience report from Siemens. In Proceedings of the International Conference on Software Engineering, 2010, DOI: 10.1145/1810295.1810300

(2) Paulisch, F. and Zimmerer, P. 2016. Collaboration of Software Architect and Test Architect Helps to Systematically Bridge Product Lifecycle Gap. In Proceedings of the 1st International Workshop on Bringing Architectural Design Thinking Into Developers' Daily Activities (Bridge), 2016, co-located workshop at International Conference on Software Engineering, 2016, DOI: 10.1145/2896935.2896936

(3) Zimmerer, P. 2016. Test Architect a key role defined. Professional Tester Magazine, issue 39, December 2016. http://www.professionaltester.com/

Agile is real – Alltag in einem XP Office

Eleonora Scherl
Volkswagen AG, Digital:Lab Berlin, Starlauer Allee 7, 10245 Berlin
nora@vw-dilab.com

Zusammenfassung

In einer sich schnell wandelnden Welt werden agile Methoden im Software Engineering allein schon aus wirtschaftlichen Gründen immer interessanter und nehmen auch in den Lehrplänen mehr Platz ein. In dem Vortrag erhalten Sie Einblick in ein Software Product Office, welches vollumfänglich mit schlanken (Lean Startup) und agilen (XP) Methoden arbeitet. Erfahren Sie wie, was und warum wir denken, dass dies das 'State of the Art' Vorgehen für modernes Software Engineering ist.

Abstract

In a rapidly changing world, agile software engineering methods are becoming more and more interesting, for economic reasons alone, as well as taking more relevance in curricula. In the talk you will get an insight into a software product office, which works completely with lean (Lean Startup) and agile (XP) methods. Learn how, what and why we think this is the 'state of the art' approach to modern software engineering to overcome uncertainty gracefully.

UML in der Hochschullehre: Eine kritische Reflexion

Jürgen Anke, Hochschule für Telekommunikation Leipzig

Stefan Bente, Technische Hochschule Köln

anke@hft-leipzig.de, stefan.bente@th-koeln.de

Zusammenfassung

Trotz ihrer hohen Bekanntheit und Verbreitung ist die Unified Modelling Language (UML) 20 Jahre nach der Vorstellung ihrer ersten Version in der Softwaretechnik nicht unumstritten. Ein wesentlicher Grund ist ihre unklare Nutzung in der Praxis die u.a. durch Ausdifferenzierung von Softwareproduktarten, agilen Entwicklungsansätzen sowie Microservices als Architekturstil mit reduzierter Komplexität beeinflusst wird. Diese Trends befördern eine Abkehr von klassischen Top-Down-Ansätzen mit sequenziellem Vorgehen, das umfangreiche Spezifikationen der Anforderungen und des Entwurfs erfordert. Für die Lehre (insbesondere auf Bachelorniveau) stellt sich daher die Frage, ob wir UML überhaupt noch brauchen und wenn ja, wofür und in welchem Umfang. Ziel dieses Beitrags ist es, empirische Belege zur Relevanz der UML zusammenzutragen und daraus Schlussfolgerungen für mögliche Kompetenzziele in der Softwaretechnikausbildung zu ziehen. Unser Ziel ist es, eine aktuelle Sicht auf die praktische Relevanz der UML zu liefern und damit die Diskussion über die Ausgestaltung der Lehre im Fach Softwaretechnik anzuregen.

Abstract

Despite its pervasiveness and popularity, the Unified Modeling Language (UML) is not uncontroversial in software engineering 20 years after the launch of its first version. A major reason is their unclear use in practice, which is influenced by the differentiation of software product types, agile development approaches and microservices as architectural style with reduced complexity. These trends move away from traditional top-down, sequential approaches that require extensive specifications of requirements and design. For teaching (especially at the Bachelor level), therefore, the question arises whether we still need UML at all and if so, for what and to what extent. The aim of this paper is to gather empirical evidence on the relevance of UML and to draw conclusions for possible competence goals in software engineering education. Our goal is to provide an up-to-date view of the practical relevance of UML to stimulate discussion about the role of UML in software engineering courses.

Einleitung und Motivation

Die Beschreibung bestehender oder geplanter Systeme ist eine wichtige Teilaufgabe im Software Engineering. Dies wird vielfach mit grafischen Modellen der Unified Modelling Language (UML) durchgeführt. Die UML ist der De-facto-Standard die grafische Modellierung von objektorientierten Systemen, um deren Analyse, Entwurf und Dokumentation zu unterstützten (Rupp, Queins, & SOPHISTen, 2012; Sommerville, 2015).

Folgerichtig ist das Erlernen der UML ein wichtiger Bestandteil in vielen Software Engineering Modulen an Hochschulen, wie es auch in den Rahmenempfehlungen der Gesellschaft für Informatik (GI) verankert ist (Gesellschaft für Informatik, 2016). Jedoch ist die UML auch in der Lehre aus unserer Erfahrung nicht unumstritten.

Aus unserer Sicht sind mögliche Gründe dafür z.B.:

- Die Relevanz der UML in der Praxis scheint zurückzugehen, da umfangreiche Spezifikationen in agilen Entwicklungsansätzen wie z.B. Scrum nicht mehr erforderlich sind.
- Aufgrund ihrer Komplexität ist die UML schwer zu erlernen (Erickson & Siau, 2007; Seiko Akayama u. a., 2013).
- Besondere Schwierigkeiten verursacht die Verbindung verschiedener Sichten auf das Softwaresystem (Boberić-Krstićev u. a., 2011).
- Klassische UML-Tools führen aufgrund ihrer Komplexität in der Lehre oft zu Herausforderungen (Liebel, Heldal, & Steghofer, 2016; Seiko Akayama u. a., 2013).
- Der Nutzen der Modellierung erschließt sich Studierenden nicht (Boberić-Krstićev u. a., 2011; Burgueño, Vallecillo, & Gogolla, 2018).

Der Eindruck einer sinkenden Bedeutung von UML wird auch anhand der Google Trends deutlich, welche die relative Häufigkeit von Suchanfragen betrachtet (Abb. 1). Hierbei wurde der Zeitraum 2004-2018 sowie die Region Deutschland gewählt. Zudem wurde mit „Themen“ statt Suchbegriffen gearbeitet, die verschiedene Formulierungen und Schreibweisen berücksichtigen.

Vor diesem Hintergrund stellt sich die Frage, ob und wie die UML in der grundständigen Software Engineering Ausbildung zu integrieren ist. Dazu haben wir uns in diesem Beitrag an folgenden Leitfragen orientiert:

- Wie und wofür wird die UML in der Praxis tatsächlich eingesetzt?
- Welche Kompetenzen sollten dafür durch die Hochschullehre entwickelt werden?

Grundlage dieser Arbeit sind empirische Studien über den Einsatz der UML anhand ihrer tatsächlichen Nutzung. Die Relevanz in der industriellen Praxis wird anhand von bestehenden Studien bewertet. Daneben haben wir eine Analyse von studentischen Projekt- und Abschlussarbeiten hinsichtlich ihrer Nutzung von grafischer Modellierung durchgeführt.

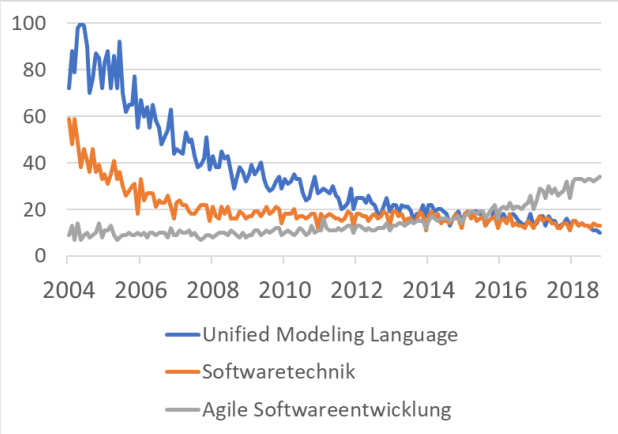


Abb. 1: Google Trends für Begriffe des Software Engineerings (2004-2018, nur Deutschland)

Dieser Beitrag ist wie folgt aufgebaut: Zunächst ordnen wir die UML historisch ein, diskutieren die Einsatzfelder sowie den Nutzen der Modellierung. Anschließend erfolgt die Analyse von Studien zur Nutzung von UML in der Praxis sowie studentischen Arbeiten. Nach der Diskussion der Ergebnisse leiten wir Schlussfolgerungen für die Rolle der UML in der Lehre ab.

UML in der Softwaretechnik

Entstehung der UML

Im Methodenkrieg der 1990er Jahre gab eine Reihe konkurrierender Ansätze für objektorientierte Methoden und Modellierungssprachen. Die erste Version der UML wurde 1997 als Vereinigung einer aus der „Unified Method“ nach Rumbaugh/Booch und dem „Object Oriented Software Engineering“ nach Jacobsen gebildet. Der Toolhersteller Rational unterstützte die UML bereits damals und trug damit zu ihrer Verbreitung bei (Rupp u. a., 2012). In den folgenden Jahren fand eine Erweiterung der UML um weitere Diagrammtypen und Konstrukte wie der Object Constraint Language (OCL) statt. Zur Standardisierung wurde die UML schließlich an die Object Management Group (OMG) übergeben. In der aktuellen Version 2.5 enthält die UML insgesamt 14 Diagrammtypen, von denen jeweils sieben zur Beschreibung der Struktur (z.B. Klassendiagramm, Verteilungsdiagramm, Paketdiagramm) und sieben zur Beschreibung des Verhaltens (z.B. Use-Case Diagramm, Aktivitätsdiagramm, Sequenzdiagramm) dienen.

Einsatzfelder und Modellarten

Modelle können beschreibend (deskriptiv) oder vorschreibend (präskriptiv) sein. Beschreibende Modelle werden einerseits für die Systemdokumentation und andererseits in der Analyse für die Beschreibung des Problemraums (Domänenmodell, Geschäftsprozesse usw.) eingesetzt. Präskriptive Modelle hingegen werden für den Entwurf und die Implementierung eingesetzt, da sie einen Bauplan („Vorschrift“) für ein zu realisierendes System beschreiben. Für die automatische Codeerzeugung auf Basis eines solchen Modells ist ein hoher Detailgrad und ein für die Zielplattform geeigneter Generator erforderlich. Wenn hingegen ein Programmierer die Implementierung manuell durchführt, kann dieser auf Basis seiner Interpretationsfähigkeiten auch mit einem weniger formalen Modell arbeiten. Tab. 1 fasst die verschiedenen Aufgaben, Modellarten und Beteiligte zusammen.

Je nachdem, wofür die UML eingesetzt wird, gibt unterschiedliche Anforderungen an die Modelle und demnach auch die Fähigkeiten der Modellersteller und -nutzer. Sommerville nennt für die Modellierung von Systemen drei verschiedene Absichten, die jeweils unterschiedliche Grade an Detaillierung und Rigorosität in der Anwendung der Modellierungssprache erfordern (Sommerville, 2015):

- *Diskussionen über ein bestehendes oder geplantes System:* Diese Modelle können unvollständig sein und die Notationen der Modellierungssprache informell verwenden.
- *Dokumentation eines bestehenden Systems:* Diese Modelle müssen ebenfalls nicht vollständig sein, wenn nur Teile des Systems dokumentiert werden. Jedoch ist Korrektheit und Genauigkeit erforderlich.
- *Codegenerierung:* Präzise Systembeschreibung als Basis für die Erzeugung der Implementation in einer modell-basierten Entwicklung.

Dies ist im Wesentlichen übereinstimmend mit der Systematisierung von Chaudon et.al., die

1. Modelle zur Analyse und Verstehen
2. Modelle zur Kommunikation
3. Modelle als Vorlage für die Implementierung
4. Modelle als Basis für die Codegenerierung

unterscheiden (Chaudron, Heijstek, & Nugroho, 2012). Die Reihenfolge dieser Nutzungsarten (von den Autoren als *modeling styles* bezeichnet) gibt ebenfalls die steigende Anforderung an Modellqualität bzw. Genauigkeit an.

Nutzen der Modellierung

Die Verwendung von UML für die Modellierung im Software Engineering soll sowohl Produktqualität als auch Entwicklungsproduktivität verbessern. Die Zwischenschritte zu diesen Wirkungen entstehen in den Kategorien Entwickler, Team, Prozess und Produkt, wie eine Studie von Chaudron u. a. (2012) ergeben hat (siehe Abb. 2). Insgesamt lässt sich sagen, dass das Ziel der Nutzung von UML im Verstehen, Beschreiben und Kommunizieren von existierenden oder geplanten Softwaresystemen in verschiedenen Lebenszyklusphasen (Analyse, Entwurf, Implementierung, Wartung) besteht. Ivar Jacobson, einer der Gründerväter von UML schreibt dazu: „Used appropriately it is a practical tool for raising the level of abstraction on software from the level of code to the level of the overall system” (Jacobson, 2009).

Die zentrale Rolle der Modellierung wird auch durch den *Guide to the Software Engineering Body of Knowledge (SWEBOK)* unterstrichen. Hier wird die UML als eine mögliche Modellierungssprache genannt und auf die Notwendigkeit verwiesen, eine für die jeweilige Aufgabenstellung geeignete grafische Notation auszuwählen (Bourque, Fairley, & IEEE Computer Society, 2014).

Phase	**Analyse**	**Entwurf**	**Implementierung**	**Dokumentation**
Original / Grundlage	Problemraum	Analysemodell	Entwurfsmodell	Implementiertes System
Modellinhalte	Analysemodelle, z.B. Domänenmodell, Geschäftsprozesse	Entwurfsmodelle, z.B. Datenstrukturen, Systemarchitektur	(Code)	Entwurfsmodell, z.B. Datenstrukturen, Systemarchitektur
Ersteller	Mensch	Mensch	Mensch	Mensch / Maschine
Nutzer	Mensch	Mensch	Mensch / Maschine	Mensch
Modellqualität	Informell, geringer Detailgrad	präzise, mittlerer Detailgrad	präzise, hohe Detaillierung	Vollständig, korrekt, genau

Tab. 1: Modellarten und ihre Eigenschaften

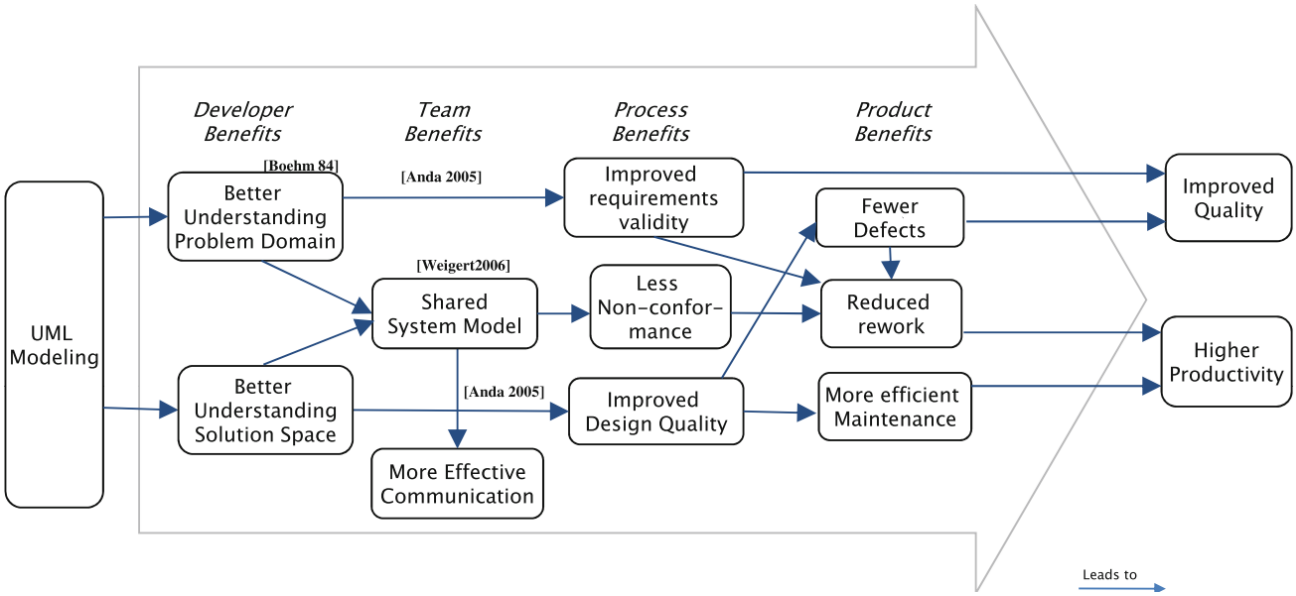


Abb. 2: Nutzen der UML Modellierung (Chaudron u. a., 2012)

UML-Nutzung durch Praktiker

Für die Beurteilung der praktischen Relevanz von UML stützen wir uns auf Studien, die den Einsatz der UML in der Praxis untersucht haben. Dabei werden einerseits die empirisch ermittelte Nutzung von UML-Modellelementen und andererseits die Verwendungszwecke untersucht.

Häufigkeit von Modellelementen der UML

Die Frage nach der Nutzung der UML anhand der tatsächlichen Verwendung von Diagrammtypen und Modellelementen wurde in den letzten 15 Jahren mittels verschiedener Methoden untersucht. Die Ergebnisse und die zugrundeliegende Methode werden nachfolgend kurz vorgestellt.

1) Delphistudie (Erickson & Siau, 2007)
   - Enterprise Systeme: 1. Klassen-, 2. Use Case-, 3. Sequenz-, 4. Aktivitätsdiagramme
   - Echtzeitsysteme: 1. Klassen-, 2. Zustands-, 3. Sequenz-, 4. Use Case-Diagramme
   - Web Applikationen: 1. Klassen-, 2. Use Case-, 3. Sequenz-, 4. Zustandsdiagramme
2) Befragung, N=284 (Dobing & Parsons, 2010)
   - 73% der Befragten nutzten in mind. 2/3 ihrer Projekte Klassendiagramme, gefolgt von Use Case (51%) und Sequenzdiagrammen (50%)
   - Klassen- und Sequenzdiagrammen wurden für die Aufgaben *Implementierung*, *Dokumentation*, *Erklären/Verständnis im Team* am häufigsten ein mindestens begrenzter Nutzen zugeschrieben.
   - Nur für *Validierung von Anforderungen mit dem Kunden* wurde der Nutzen bei Aktivitäts- und Use Case-Diagrammen am höchsten eingeschätzt.
   - Der Einbezug von Kunden wurde anhand ihrer Mitwirkung beim *Entwickeln*, *Prüfen* und *Freigeben* von Modellen bewertet. In allen Fällen wird das Use Case Diagramm am häufigsten genannt, gefolgt vom Aktivitätsdiagramm.
3) Analyse von Lehrbüchern, UML Tutorials, Hochschulkursen (Gianna Reggio, Maurizio Leotta, Filippo Ricca, & Diego Clerissi, 2013)
   - In Hochschulkursen sind Klassen-, Sequenz-, Aktivitäts-, Use Case- und Zustandsdiagramme besonders häufig.
   - In UML Tutorials zusätzlich dazu Komponenten- und Verteilungsdiagramme.
   - Die am seltensten genutzte Diagrammtypen sind Timing-, Profil-, Interaktionsübersichts- und Kompositionsstrukturdiagramme.
4) Quantitative Analyse von Open UML Modellen, N=121 (Langer, Mayerhofer, Wimmer, & Kappel, 2014)
   - Häufigste Sprachelemente: Class (100%), Use Case (47%), Interactions (39%)
   - Häufigste Elemente in Interaktionsdiagrammen: Message, Lifeline
   - Profile werden häufig für Robustness-Diagramme u. Datenbankschemata genutzt
   - Klassen sind die am häufigsten mit Stereotypen versehenen Modellelemente.

5) Experteninterviews, N=50 (Petre, 2013)
   - 35 von 50 Befragten nutzen UML gar nicht, v.a. wegen des Fokus auf die Softwarearchitektur und nicht auf das ganze System (fehlender Kontext), dem hohen Aufwand zum Verstehen der UML (unnötige Komplexität) sowie Problemen von Synchronisierung bzw. Inkonsistenzen zwischen Modell und Code bzw. Modellsichten
   - 11 von 50 Befragten nutzen UML nur selektiv, besonders in frühen Phasen zur Bewertung von Ideen, auch im Team. Sie weniger verwendeten UML meist in kleineren Modellen, adaptierten die Notation an ihre Bedürfnisse und beendeten die Verwendung, wenn UML in der jeweiligen Projektphase nicht mehr gebraucht wurde.
   - 3 der 50 Befragten nutzten UML für die automatische Codeerzeugung. Die entsprechenden Einsatzfälle waren Software für eingebettete System oder Produktlinien.

Einsatz von konzeptioneller Modellierung

Bei diesem Aspekt geht es um die Erhebung von Einsatzfällen für Modellierung, z.B. in Anhängigkeit von Unternehmensgrößen, Modellierungserfahrung, Projekttyp oder Entwicklungsphase. Die hier betrachtete konzeptionelle Modellierung umfasst auch andere Sprachen, nicht nur die UML. Wie auch schon bei den Studien zur Häufigkeit von Modellelementen werden hierfür unterschiedliche Forschungsmethoden eingesetzt.

1) Befragung zur Modellierung, N=312 (Davies, Green, Rosemann, Indulska, & Gallo, 2006)
   - Die häufigsten regelmäßig eingesetzten Modellierungstechniken sind ER-Diagramm (42%), Datenflussdiagramm (34%) und Flowcharts (29%). UML ist mit 21% auf Platz 5.
   - Die meistgenutzten Tools sind MS Visio (44%) und Rational Rose (11%).
   - Unternehmensgröße: Die UML wird bei mittleren Unternehmen (100-1000 Mitarbeiter) häufiger eingesetzt als bei kleineren oder größeren Unternehmen.
   - Modellierungserfahrung: Mitarbeiter mit 4-10 Jahren Erfahrung nutzen häufiger konzeptionelle Modelle als Kollegen, die kürzere oder längere Erfahrung haben.
   - Die häufigsten Einsatzzwecke für konzeptionelle Modellierung sind (1) Datenbankdesign/–management, (2) Geschäftsprozessdokumentation, (3) Interne Prozessverbesserung, (4) Softwareentwicklung, (5) Verbesserung kollaborativer Prozesse.
2) Analyse der Repositories von Open Source Projekten, N=10 (Osman & Chaudron, 2013)
   - Das Verhältnis von modellierten Klassen und implementierten Klassen lag zwischen 4% im größten Projekt (~900 Klassen) und 40% im kleinsten Projekt (<60 Klassen)
   - Nur wenige Projekte aktualisierten ihre initialen Modelle, meist wurden bei neuen Features neue Modelle hinzugefügt.
3) Befragung zum Einsatz von Entwurfsmodellen, N=3785 (Gorschek, Tempero, & Angelis, 2014)
   - Entwurfsmodelle werden in der Praxis nicht häufig verwendet.
   - Modelle werden eher informell und ohne Toolsupport erstellt.
   - Die Notation ist tendenziell nicht UML.
   - Die Nutzung von Modellen nimmt mit Grad an Erfahrung und Qualifikation ab
   - Modelle werden eher zur Kommunikation und Zusammenarbeit genutzt und auf Whiteboard oder Papier gezeichnet und werden nach Erstellung selten aktualisiert.
4) Befragung zum Einsatz von Skizzen und Diagrammen im Software Engineering, N=394 (Baltes & Diehl, 2014)
   - 48% aller Skizzen enthalten einige UML-Elemente, 9% ausschließlich UML.
   - Die Hälfte der Skizzen wurde von einer einzelnen Person angefertigt, 28% von zwei und 15% von drei Personen.
   - 58% aller Skizzen wurde analog (Papier, Whiteboard) angefertigt, 40% digital.
   - Digitale Skizzen werden im Vergleich zu analogen Skizzen zu einem höheren Teil (77%) überarbeitet und archiviert (94%).
   - Wesentliche Einsatzzwecke sind Entwerfen (75%), Erklären (65%), Verstehen (56%) sowie Anforderungen analysieren (45%).

Zwischenfazit und Diskussion

Da einige Studien z.T. schon über 10 Jahre alt sind, können die Ergebnisse der Studien nur mit Vorsicht auf die heutige Verwendung extrapoliert werden. Dennoch lassen sich gewisse Tendenzen bei der Nutzung von UML (bzw. konzeptionellen Modellierungsansätzen insgesamt) in der Praxis der Softwareentwicklung durchaus ausmachen.

Tendenzen in der UML-Nutzung

Die UML hat gemäß den ausgewerteten Studien in der Praxis eine sehr hohe Bekanntheit und auch eine hohe Nutzung. Allerdings werden in der Praxis die einzelnen Diagrammtypen unterschiedlich häufig verwendet. Insbesondere Klassen-, Aktivitäts-, Use-Case- und Sequenzdiagramme sind verbreitet.

Die Verwendungszwecke sind oft in frühen Phasen der Systementwicklung, z.B. zur Unterstützung der Analyse. Dabei spielt Modellierung besonders für die Kommunikation und Zusammenarbeit zwischen den Beteiligten eine wichtige Rolle.

Modelle sind häufig informell oder werden mit informellen Notationen gemischt. Alternativen zur Modellierung mit UML sind weniger formelle Modellierungsansätze wie das C4-Architekturmodell (Brown, 2018), textuelle Domain Specific Languages (DSL) oder auch Ad-Hoc-Notationen.

Als Medien werden neben Papier und Whiteboard auch verschiedene Tools eingesetzt, wobei diese nicht notwendigerweise klassische UML-Modellierungstools sind. Wenn ein konsistentes Gesamtmodell mit vielen untereinander verbundenen Sichten weniger wichtig ist, kann auf ein Model Repository verzichtet werden. In diesen Fällen können auch Zeichentools wie Visio oder eines der zahlreichen auf Kollaboration ausgelegten webbasierten Zeichenwerkzeuge für die UML-Modellierung genutzt werden.

Die Erzeugung von Code aus UML-Modellen spielt in der Praxis eine sehr untergeordnete Rolle. Der Ansatz der modellgetriebenen Architektur (MDA) mit umfangreicher Codeerzeugung aus detaillierten UML-Modellen hat sich im Wesentlichen nur in für langlebige Systeme in gut verstandenen Domänen etabliert (Sommerville, 2015).

Gründe für die veränderte Nutzung von UML

Trotz ihrer Verbreitung wird die UML in der Praxis in sehr unterschiedlichem Maße eingesetzt. Hierfür lassen sich eine Reihe von Gründen identifizieren.

Die zunehmende Diversifizierung von Softwareproduktarten schränkt der Nutzen einer einheitlichen Modellierungssprache ein. Die Diversität der Softwarelandschaft insgesamt nimmt zu. Beispiele sind die weite Verbreitung von Webapplikationen und mobilen Apps oder das Nebeneinander von Endkundenorientierten E-Commerce-Applikationen neben klassischen betrieblichen Informationssystemen oder sicherheitskritischen Echtzeitsystemen. Diese verschiedenen Softwareprodukte unterscheiden sich sowohl in ihren technischen Plattformen wie auch in ihren typischen Projektkonstellationen stark voneinander. Dies betrifft u.a. die Stabilität der Anforderungen, die Größe des Projektteams sowie den formalem Projektrahmen, was wiederum den Nutzen der Modellierung beeinflusst.

Agile Entwicklung braucht weniger UML. Agile Methoden basieren auf iterativer und inkrementeller Entwicklung, in denen die Phasen Analyse, Entwurf, Implementierung, Test und Dokumentation in kurzen Zyklen wiederholt werden. Dadurch sinkt die Notwendigkeit einer umfangreichen Spezifikation des gesamten Systems bzw. großer Systemteile, wofür in einem klassischen Wasserfall-Vorgehen häufig UML eingesetzt wird.

In der agilen Welt wird die Rolle des Software-Architekten, die klassischerweise UML als wesentliches Spezifikations- und Kommunikationsinstrument verwendet, mit einer gewissen Skepsis betrachtet. Ein Beleg ist etwa das Prinzip 11 des Agilen Manifests: "The best architectures (…) and designs emerge from self-organizing teams", (Beck u. a., 2001), kritische Aussagen zu „klassischer" Softwarearchitektur finden sich darüber hinaus an vielen Stellen in der agilen Literatur (z.B. Coplien & Bjørnvig, 2010, S. 85). Diese Grundhaltung trägt dazu bei, dass formale Modellierung in agilen Projekten weniger verbreitet ist, auch wenn seit mehreren Jahren eine Reihe von Frameworks existieren, die agiles Vorgehen mit Architekturplanung verbinden (siehe z.B. Cockburn, 2006; Larman & Vodde, 2009; Leffingwell, 2007).

Microservices haben keine zentralen Modelle mehr. Das Aufkommen des Microservice-Architekturstils als Alternative zur monolithischen oder service-orientierten Architektur ist eng verbunden mit der agilen Vorgehensweise. Microservices sind so weit wie möglich lose gekoppelt, unter Inkaufnahme von Daten-Redundanzen. Eine klassische Top-Down-Modellierung von Applikations-, Daten- und Technologiearchitektur findet hier nicht mehr statt (oder in einer wesentlich abstrakteren, Prinzipien-orientierten Weise, die nicht unbedingt mehr eine formale Modellierung erfordert).

Die Services selbst sind eher klein und interagieren mit ihrer Umgebung häufig durch REST- und Eventschnittstellen, die sich ebenfalls gut ohne eine graphische Modellierung spezifizieren lassen. Diese maximal autarken Softwareeinheiten ermöglichen eine idealtypische Anwendung des agilen Ansatzes selbstorganisierter, autarker und im Wesentlichen auf mündliche (oder zumindest informelle) Kommunikation ausgerichtete Entwicklungsteams. Eine Modellierungssprache wie UML, die inhärent von einem zentralen Gesamtmodell mit vielen komplementären Sichten ausgeht, stiftet in diesem Kontext nur geringen Nutzen.

Domain Driven Design modelliert fachlich und weniger komplex. Domain Driven Design (Evans, 2003) ist der dem Microservice-Stil zugrundeliegende verbundene Entwurfsansatz. Der Designfokus liegt hier auf dem Verständnis und der Modellierung der Domäne, also der Fachlichkeit einer Anwendung. Die Gestaltung der Applikations- und Technologiearchitektur rückt in Hintergrund, gemäß der Überzeugung, dass diese nachgeordnet und dezentral gestaltet wird. Domain Driven Design nutzt zur Analyse nicht-graphische Begrifflichkeiten wie die Ubiquitous Language oder informelle Modellierungsformen wie Context Maps (Fowler, 2014), die nicht standardisiert sind und sich höchstens (im Fall der Context Map) grob an UML-Klassendiagramme anlehnen.

UML-Nutzung durch Studierende

Wir haben anhand von 47 Abschlussarbeiten und Projektberichten in den Studiengängen Informatik, Medien- und Wirtschaftsinformatik (Bachelor und Master) der TH Köln untersucht, ob und welche Modellierungsansätze Studierende verwendet haben. Es wurden nur Arbeiten betrachtet, in denen die Konzeption und/oder Erstellung eines Softwaresystems mindestens einen Teilschwerpunkt darstellte. Die Arbeiten durften nicht älter als fünf Jahre sein, um ein aktuelles Bild zu gewinnen.

Weiterhin durfte es keinerlei Vorgaben seitens der Betreuer geben, ob (und wenn ja welche) Modellierungsansätze zu verwenden waren[1]. Da die betrachteten Arbeiten also UML nicht explizit erfordern, lässt sich aus den Ergebnissen eine Tendenz ableiten, inwiefern Studierende UML (oder konkurrierende Ansätze) als Mehrwert bei der Spezifikation und Dokumentation empfinden.

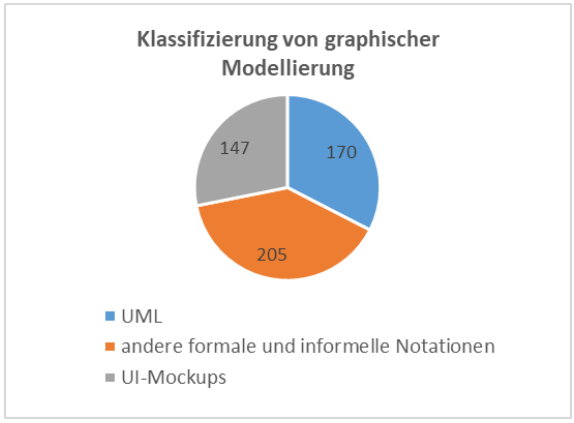


Abb. 3: Modellierungsnutzung durch Studierende

Wie Abb. 3 zeigt, verwenden weniger als die Hälfte der graphischen Modellierungen in den untersuchten Arbeiten UML. In knapp der Hälfte der Arbeiten werden zudem UI-Mockups genutzt, die im Folgenden jedoch nicht weiter betrachtet werden.

In den analysierten Arbeiten spielen nur Klassen-, Komponenten-, Aktivitäts-, Use-Case- und Sequenzdiagramme eine nennenswerte Rolle (siehe Abb. 4). In jeweils einer Arbeit kommt auch ein Objekt- und Deployment-Diagramm zum Einsatz. Das mag daran liegen, dass diese Diagrammtypen gerade diejenigen sind, die in den Softwaretechnik-Vorlesungen in Informatik, Medien- und Wirtschaftsinformatik vermittelt werden (nur Zustandsdiagramme fallen aus der Reihe, die zwar gelehrt, aber in den Arbeiten nicht auftreten). Davon abgesehen bestätigen sich die Ergebnisse der oben zitierten Studien zum UML-Einsatz: Eine Teilmenge der UML-Diagramme wird in der Praxis in nennenswertem Umfang verwendet.

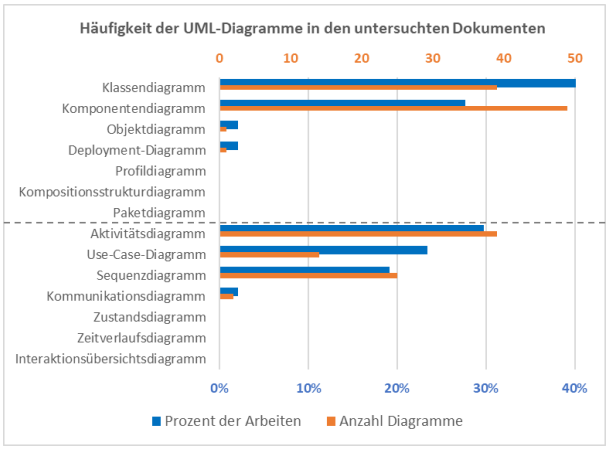


Abb. 4: Häufigkeit von UML-Diagrammen

Eine weitere Erkenntnis lässt sich aus Abb. 4 ableiten: Komponentendiagramme treten in knapp 30% der Arbeiten auf, dort gibt es davon dann insgesamt 49 Instanzen. Klassen- und Use-Case-Diagramme werden in ähnlich vielen Arbeiten verwendet, aber mit weniger Instanzen. Wenn also eine Arbeit Komponentendiagramme enthält, dann in der Regel mehrere; Use-Case-Diagramme hingegen treten fast überall maximal einmal auf, bei Klassendiagrammen ist es ähnlich.

Einsatzkontexte graphischer Modellierung

Ein wesentliches Ziel der Untersuchung war es, die Verwendung von UML-Diagrammen mit der von Nicht-UML-Notationen zu vergleichen. Hierfür wurden zwölf Einsatzkontexte für graphische Modellierung definiert, in denen neben UML auch andere formale Notationen sowie informelle Darstellungen in den untersuchten Arbeiten auftraten. Diese sind in Tab. 2 übersichtsweise dargestellt.

Die Häufigkeit des Auftretens der verschiedenen Einsatzkontexte (unabhängig davon, ob UML oder eine alternative Darstellung gewählt wurde) ist in Abb. 5 dargestellt. Demnach wird eine Komponentenstruktur in über der Hälfte der Arbeiten graphisch modelliert, technisch-algorithmische Abläufe, eine Klassenstruktur der Implementierung sowie eine Übersicht der Anwendungsfälle sind ebenfalls oft anzutreffen.

[1] Dadurch schieden z.B. Praktikumsberichte in Veranstaltungen aus, bei denen die Beschäftigung mit UML-Modellierung ein explizites Lernziel war. Die untersuchten Arbeiten wurden von neun verschiedenen Professoren betreut. Damit dürfte sich auch der Einfluss von dem Bemühen der Studierenden, ihrem Betreuer in dessen persönlichen Modellierungsvorlieben zu gefallen, über die Gesamtheit der Ergebnisse verwischen.

Einsatzkontext	**UML-Diagramm**	**Andere formale Notation**	**Informelle Variante**
Anwendungsfälle	Use Case	ArchiMate Business Services	Verknüpfung von Aktoren und Anwendungsfällen
Nutzungsszenario / Geschäftsprozess	Aktivitätsdiagramm	BPMN, EPK	Informelles Flussdiagramm
Fachliche Zustände von Geschäftsobjekten	Zustandsdiagramm	-	Informeller Graph
Fachliches Datenmodell	Klassendiagramm	-	Informeller Graph
Technischer Prozess / algorithmischer Ablauf	Aktivitätsdiagramm	BPMN, EPK	Informelles Flussdiagramm
Technische Zustandsfolge (State Machine)	Zustandsdiagramm	-	Informeller Graph
Technisches Kommunikationsszenario	Sequenz- oder Kommunikationsdiagramm	-	Mit Pfeilen und Sequenz-Zahlen annotierte Strukturen
Logisches Datenmodell / Domänenmodell	Klassendiagramm	ER-Diagramm	Informeller Graph
Physisches Datenmodell	Klassendiagramm	ER-Diagramm	Informeller Graph
Klassenstruktur der Implementierung	Klassendiagramm	-	Informeller Graph, teilw. aus Technniksymbolen
Komponentenstruktur	Komponentendiagramm	-	Informelle Block- oder Symbolgraphik
Verteilungssicht	Deploymentdiagramm	-	Informelle Block- oder Symbolgraphik

Tab. 2: Untersuchte Einsatzkontexte graphischer Modellierung

Allerdings ist es etwas ernüchternd, dass im Median jede untersuchte Arbeit nur jeweils zwei der genannten Einsatzkontexte graphisch modelliert, mit jeweils vier Diagrammen. Abb. 6 zeigt die Häufigkeit der Wertepaare „(Anzahl Diagramme, Anzahl Einsatzkontexte)“ als Blasendiagramm (je häufiger ein Wertepaar auftritt, desto größer die Blase). Bei Diagrammen liegt der Mittelwert mit ca. acht Diagrammen je Arbeit deutlich über dem Median. Einzelne Arbeiten verwenden also graphische Modellierung deutlich größerem Ausmaß als der Rest.

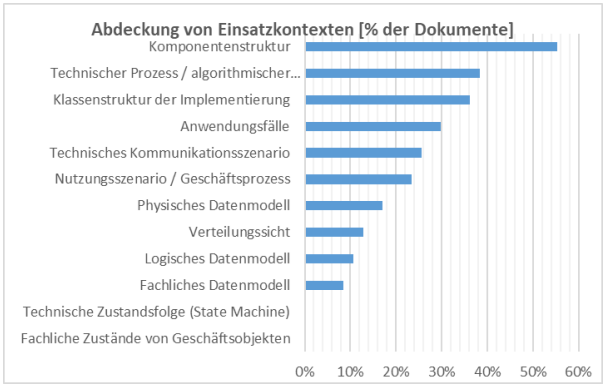


Abb. 5: Abdeckung der Einsatzkontexte

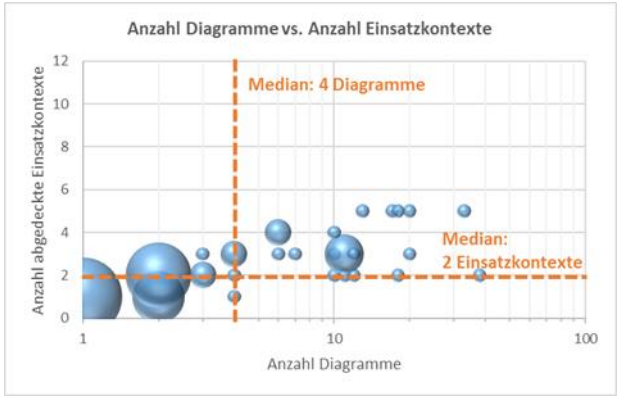


Abb. 6: Anzahl Diagramme vs. Einsatzkontexte

Verwendung von UML im Vergleich mit anderen Notationen

UML-Klassendiagramme stellen in gleich vier von zwölf Einsatzkontexten eine der Alternativen dar (siehe Abb. 7). Am häufigsten wird die Klassenstruktur der Implementierung dargestellt, hier trifft man hauptsächlich auf UML-Klassendiagramme. Das physische Datenmodell wurde in den untersuchten Arbeiten hingegen ausschließlich als ER-Diagramm modelliert. Logisches und fachliches Datenmodell werden nur selten modelliert.

UML-Aktivitätsdiagramme sind in zwei der zwölf untersuchten Einsatzkontexte von Belang. In beiden tritt eine UML-Modellierung gleichberechtigt neben anderen Darstellungsformen auf (siehe Abb. 8). Bei technischen Prozessen und algorithmischen

Abläufen trifft man häufig eine informelle Flussdiagramm-Notation an, während Geschäftsprozesse oder Nutzungsszenarien häufig als BPMN (und gelegentlich auch als EPK) modelliert waren.

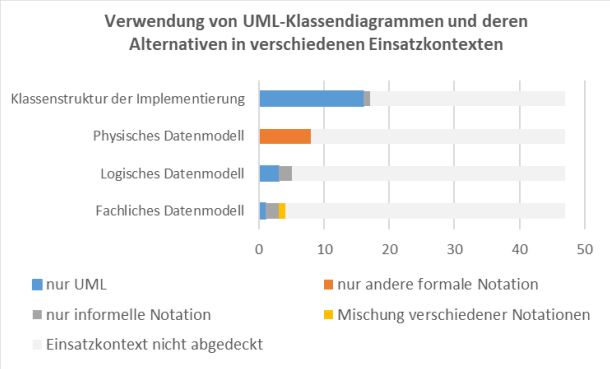


Abb. 7: Einsatzkontexte mit UML-Klassendiagrammen

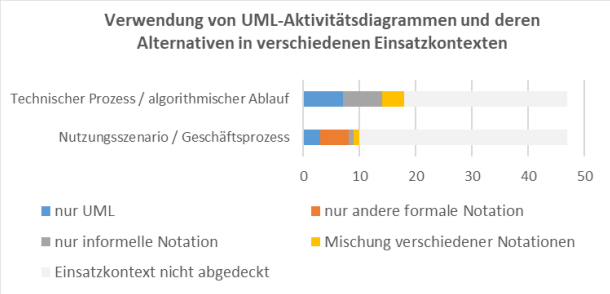


Abb. 8: Einsatzkontexte mit UML-Aktivitätsdiagrammen

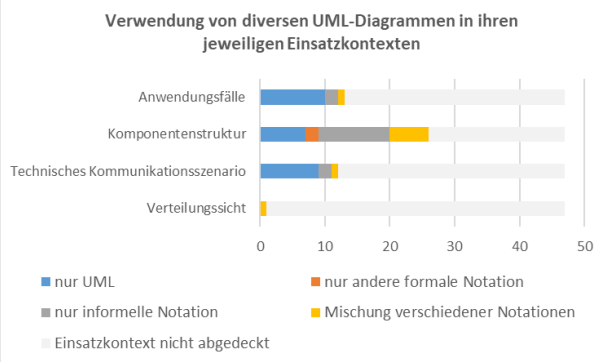


Abb. 9: Verwendung von weiteren UML-Diagrammen in ihren jeweiligen Einsatzkontexten

In Abb. 9 sind die Einsatzkontexte dargestellt, in denen UML-Use-Case-, Komponenten-, Sequenz-, Deployment- und Kommunikationsdiagramme eine Rolle spielen. Komponentendiagramme haben in einer informellen Darstellung mit gestapelten oder geschachtelten Blöcken eine große Konkurrenz. Anwendungsfälle werden hingegen in der Regel als Use-Case-Diagramme modelliert. Gleiches gilt für Kommunikationsszenarien, hier dominieren Sequenz- und Kommunikationsdiagramme.

Zwischenfazit und Diskussion

Auch wenn geplant ist, die Untersuchung noch weiterzuführen und ihre Datenbasis zu verbreitern, lassen sich schon einige Erkenntnisse festhalten.

Graphische Modellierung wird benötigt. Graphische Modellierung hat ihren festen Platz in der Dokumentation studentischer Softwarekonzeption und -Implementation. Allerdings streuen die untersuchten Arbeiten stark in Umfang und Konsistenz der verwendeten Modellierungsansätze. Möglicherweise liegt dies einfach an der natürlichen Qualitätsverteilung der untersuchten Arbeiten.

Nur ein Subset der UML-Spezifikation ist für studentische Arbeiten relevant. Die Untersuchungen zeigen, dass in den Arbeiten im Wesentlichen diejenigen Diagramme verwendet werden, auf die sich auch die Lehre an der TH Köln beschränkt (und die auch in den Praxisstudien als die dominierenden Typen festgestellt werden). Ob die Studierenden einfach das einsetzen, was sie im Studium kennengelernt haben, oder aber diese Diagrammtypen anscheinend ihren Zweck gut erfüllen, lässt sich aus dieser Untersuchung nicht klären. Die Tatsache, dass so viele andere formale und informelle Notationen eingesetzt werden, spricht doch für eine „Freiwilligkeit". Sprich: Es ist zu vermuten, dass die Studierenden nur Diagrammtypen verwenden, die sie für sinnvoll erachten und gut verstehen.

Informelle Notationen haben ihren Platz. Es fällt auf, dass informelle Notationen oft eingesetzt werden, in denen UML möglicherweise eine Überformalisierung darstellt (Komponentenstrukturen, technisch-algorithmische Abläufe). Dort, wo die UML-Diagramme jedoch entweder intuitiv klar (Use-Case-Diagramme) und/oder kompakt und ausdrucksstark (Klassen-, Sequenz-Diagramme) sind, kommen informelle Alternativen selten vor.

Andere formale Notationen haben ebenfalls ihren Platz. Bei Geschäftsprozessen und physischem Datenmodell dominieren andere Notationen als UML. Möglicherweise liegt dies an der Struktur der Informatik-Ausbildung an der TH Köln, die in den entsprechenden Fächern (Datenbanken, Informationsmanagement) die Standards „ER" und „BPMN" lehrt. Vielleicht sind diese Standards aber einfach tatsächlich verbreiteter als ihre UML-Gegenstücke.

Das Potential graphischer Modellierung für fachliche Aspekte wird nicht ausgeschöpft. Betrachtet man die Ergebnisse in Abb. 5, so sind die drei am häufigsten abgedeckten Einsatzkontexte graphischer Modellierung allesamt technischer Natur. Die fachlich geprägten Kontexte „Anwendungsfälle", „Nutzungsszenario / Geschäftsprozess" und „fachliches Datenmodell" stehen auf den Plätzen 4, 6 und 10 (von 12). Klassendiagramme werden überwiegend für Code-Dokumentation verwendet (Abb. 7). Zu-

standsfolgen von Geschäftsobjekten (die ein starkes Werkzeug fachlicher Modellierung sind) wurden in den untersuchten Arbeiten gar nicht gefunden. Dies alles zeigt ein deutlich technisches Übergewicht der graphischen Modellierung (ob in UML oder in anderer Form). Auf das Potential fachlicher Modellierung sollte also in der Lehre noch stärker hingewiesen werden.

UML in der Lehre – belassen, verändern, abschaffen?

Die Beschreibung von Software mithilfe von Modellen ist eine essentielle Aufgabe und gehört daher zum Kernbestandteil der Softwaretechnikausbildung. Bereits von 10 Jahren stellte Glinz dazu ein Thesenpapier über die Rolle der Modellierung in der Lehre vor (Glinz, 2008), das jedoch nicht spezifisch auf UML bezogen ist.

Die dargestellten empirischen Untersuchungen zeigen, dass die UML in der Praxis eher selektiv eingesetzt wird. Eine eigene schlaglichtartige Analyse von studentischen Arbeiten bestätigt diesen Eindruck auch für diejenigen Informatikern, die gerade ins Berufsleben eintreten.

Welcher Schluss ist daraus für die zukünftige Ausgestaltung und Weiterentwicklung der Softwaretechnik-Lehre zu ziehen? Ist die UML-Ausbildung in ihrer jetzigen Form noch zeitgemäß? Sollte sie verändert werden? Oder wäre es sinnvoll, sie ganz aus den Curricula von Informatik-Studiengängen zu verbannen? Diese Fragen kann nur durch eine Diskussion in der Fachcommunity beantwortet werden. Dazu möchten wir nicht nur die möglichen Alternativen aufzeigen, sondern auch selbst Stellung beziehen.

Die Erkenntnisse und Schlussfolgerungen aus Recherche und eigenen Erhebungen legen unserer Meinung nach nahe, dass eine UML-Ausbildung in der Form, wie sie vor 15 Jahren angemessen war, heute nicht mehr in die Zeit passt. Die Gründe hierfür haben wir unter „Zwischenfazit und Diskussion“ bei UML-Nutzung durch Praktiker und durch Studierende aufgezeigt. Ein reines „Belassen“ ist unserer Ansicht nach also keine sinnvolle Option.

Ebenso wenig erscheint es angebracht, UML komplett aus den Curricula zu entfernen. Unsere Erkenntnisse legen nicht den Schluss nahe, dass UML für die Softwaretechnik nicht mehr relevant ist.

Damit erscheint eine Anpassung des jetzigen Lehrkonzepts der sinnvollste Ansatz zu sein. Aus den obigen Analysen und Erkenntnissen lassen sich durchaus Bereiche identifizieren, in denen eine Neujustierung der Lehre sinnvoll ist. Um eine Fachdiskussion zum Thema anzuregen, haben wir unsere Vorschläge als Thesen strukturiert.

These 1. UML-Kompetenz sollte in der Softwaretechnikausbildung weiterhin vermittelt werden.

Die hohe Verbreitung von UML und ihre Nutzung als mittlere Abstraktionsebene zwischen fachlichem Problem und Implementierung lassen ihre Behandlung in der Lehre weiterhin sinnvoll erscheinen.

Die Aktivität der modellbildenden Abstraktion ist eine grundlegende Querschnittskompetenz in der Informatik (Engels, Hausmann, Lohmann, & Sauer, 2006). Vor diesem Hintergrund dient die Beschäftigung mit UML grundsätzlich dem Erlernen dieser Kompetenz – ganz unabhängig davon, ob man die konkreten Modellelemente auch tatsächlich später in seiner beruflichen Praxis verwendet.

Darüber hinaus scheinen einige UML-Diagrammtypen intuitiv verständlich und gut anwendbar zu sein, oder zumindest als bewusste oder unbewusste Vorlagen für eine informelle Modellierung zu dienen. Sie können daher als eine Art von informatischer Allgemeinbildung angesehen werden. (siehe auch These 2.)

These 2. Die Lehre sollte sich auf ausgewählte UML-Modellelemente fokussieren.

Die dargestellten Studienergebnisse legen nahe, dass ein Anspruch an Studierende, die UML-Sprachelemente möglichst umfänglich zu beherrschen, nicht sinnvoll ist. Insbesondere die hohe Komplexität und der hohe Formalisierungsgrad der UML erscheint nicht mehr zeitgemäß, da insbesondere die Maschinenlesbarkeit von UML Modellen für Codegenerierung bzw. direkter Ausführung („Executable UML“) sich nicht in der Breite durchsetzen konnten.

Wie die empirischen Untersuchungen zeigen, werden UML-Modelle vielfach informell zur Kommunikation und Zusammenarbeit eingesetzt. In diesen Situationen werden nur wenige Sprachelemente benötigt. Einige Diagrammtypen scheinen intuitiv verständlich und gut merkbar zu sein. Sowohl in der Praxis wie auch in der Auswertung studentischer Arbeiten werden diese Elemente mehr oder weniger nahe am definierten Standard verwendet. Dies trifft insbesondere auf Klassen-, Use-Case und Sequenz-Diagramme zu.

Andere UML-Diagrammtypen findet man häufig auch in einer informellen, „verballhornten“ Form, die sich aber durchaus mehr oder weniger deutlich an den UML-Sprachelementen orientiert. Beispiele hierfür sind Komponenten-, Deployment-, Aktivitäts-, Zustands- und Kommunikationsdiagramme. Diese Tatsache allein ist unserer Ansicht nach Grund genug, Kompetenzen in den „originalen“ UML-Diagrammtypen in der Lehre zu vermitteln.

Eine größere Bedeutung hat die UML nach wie vor für die Analyse, z.B. für Domänen- oder Geschäfts-

prozessmodelle. Da in diesen Fällen sowohl Modellersteller als auch -nutzer Menschen sind, ist ein geringeres Maß an Formalität ausreichend.

Dafür wird der durch die hohe Ausdrucksstärke bedingte große Umfang an Modellelementen jedoch nicht benötigt. Dies sorgt zum einen für Konkurrenz durch informelle Modellierungsansätze, für eine hohe Hürde beim Erlernen der UML sowie auch mangelhafte Einsicht in die Notwendigkeit der Sprachbeherrschung bei den Studierenden.

Konkrete Kandidaten für eine Verschlankung in der Lehre sind nach unserer Ansicht die Vielzahl von Beziehungstypen in Komponentendiagrammen, detaillierte Spezifikation von Extension Points in Use-Case-Diagrammen oder komplexe Entry- und Exit- Aktionen in Zustandsdiagrammen.

These 3. Die Lehre sollte sich auf die Beherrschung von Einsatzkontexten anstatt von UML-Diagrammtypen fokussieren.

Die GI gibt für die Informatikausbildung an Hochschulen folgende inhaltliche Empfehlungen mit Bezug zur UML (Gesellschaft für Informatik, 2016):

- *Stufe 1 (Verstehen):* „Verschiedene Notationen wie z.B. UML für die Modellierung von Softwaresysteme erläutern."
- *Stufe 2 (Anwenden):* „Standardsituationen im Bereich der Modellierung (…) umsetzen. Die Begriffswelt des Anwenders durch geeignete Vorgehensweisen erfassen und zu einer fachlichen Terminologie im Projekt verdichten."
- *Stufe 3 (Analysieren):* „Die Eignung eines Vorgehensmodells, einer Notation oder einer Methode für ein klassifiziertes Softwaresystem oder eine klassifizierte Aufgabe einschätzen."

In dieser Systematik ist 2 (Anwenden) die praxisrelevanteste Lernstufe. Allerdings wird in realen Projekten nicht die Kompetenz „Entwerfe ein UML-Klassendiagramm" gefordert, sondern „Entwerfe ein fachliches oder logisches Datenmodell". Daher plädieren wir dafür, in der Formulierung von Lernzielen den Fokus von UML-Diagrammtypen hin zu Einsatzkontexten graphischer Modellierung zu lenken. In Tab. 2 haben wir dazu einen Vorschlag von zwölf Einsatzkontexten gemacht.

These 4. Die Fähigkeit zur Auswahl von Modellierungsansätzen sollte gestärkt werden.

Neben der UML haben sich diverse andere mehr oder weniger formale Modellierungsansätze etabliert. Informelle Modellierungsansätze sowie formale Nicht-UML-Modellierungssprachen sollten daher vergleichend in die Lehre einfließen. Studierende sollten die Kompetenz erwerben, geeignete Modellierungsansätze, Diagrammtypen und den angemessenen Detailgrad im Modell in Abhängigkeit von der Aufgabenstellung auszuwählen. Dies ist konform mit Stufe 3 der o.g. GI-Empfehlung.

These 5. Die Modellierungskompetenz sollte auf das Absolventenprofil abgestimmt sein.

Sinnvoll wäre aus unserer Sicht eine Herleitung der angestrebten Modellierungskompetenz anhand des gewünschten Kompetenzprofils pro Studienfach und bestehenden Vorkenntnissen. Bei reinen Informatikstudiengängen ist in der Regel das Berufsbild des Softwarearchitekten Teil des Absolventenprofils, was einen bewussten Umgang mit Metaebenen und Abstraktion voraussetzt. Im Gegensatz dazu steht bei angehenden Medien- oder Wirtschaftsinformatikern eher die Anwendung im Vordergrund. Verschiedene technische Studiengänge sind z.B. mit der Entwicklung von Echtzeitsystemen befasst, in denen formale Verifikation und Codegenerierung relevant sind.

Zudem spielt Ausrichtung des Studiengangs auf eine Zieldomäne eine wichtige Rolle. Dabei sollten die verschiedenen Schwerpunkte in Softwareprodukten (Web vs. Backend, E-Commerce / Social Media vs. betriebliche Anwendungssysteme) sowie Projektmanagementansätze (agil vs. phasenorientiert) als Leitlinie dienen. In weiterführenden Modulen zum Thema Softwareproduktlinien oder Softwarewartung können besondere Einsatzbereiche der Modellierung erschlossen werden.

These 6. Die UML-Ausbildung sollte die Modellierung von Fachlichkeit stärker fokussieren.

Die Lehre sollte die frühen Phasen der Modellbearbeitung stärker in den Fokus nehmen. UML hat große Stärken in der Beschreibung einer fachlichen Domäne in Form eines fachlichen Datenmodells sowie von geschäftlichen Abläufen als Aktivitätsdiagrammen. Transaktionale Vorgänge lassen sich sehr gut als Zustandsdiagramme eines Geschäftsobjekts beschreiben.

Unsere Auswertung der studentischen Arbeiten hat im Gegensatz dazu gezeigt, dass UML eher für die technische Beschreibung aus Entwickler- statt aus Nutzersicht eingesetzt wird (siehe z.B. Abb. 7).

These 7. Die Lehre sollte nicht Tools, sondern Kommunikation und Interaktivität in den Fokus stellen.

Softwareentwicklung wird immer stärker durch Kommunikation und Zusammenarbeit geprägt. Damit sollte die Ausbildung die interaktive Nutzung von UML stärker einüben, z.B. in der Analyse- und frühen Entwurfsphase, in der in der Praxis oft auf Papier oder am Whiteboard gearbeitet wird.

Für die praktische Umsetzung ergeben sich daraus interessante Fragestellungen in Bezug auf die konkrete didaktische Umsetzung sowie die damit verbundene geeignete Toolunterstützung.

Zusammenfassung und Ausblick

Die Bedeutung von UML für die Softwaretechnik ist in Veränderung begriffen. Die Ursachen dafür liegen in Trends, die umfangreiche Modellierungen nicht mehr erforderlich machen. Dazu gehören agile Entwicklungsansätze, Microservicearchitekturen sowie das weitgehende Scheitern des MDA-Ansatzes, der detaillierte und formale Modelle erfordert. Die Aufmerksamkeit, die diesen Themen derzeit geschenkt wird, darf nicht darüber hinwegtäuschen, dass in vielen Großprojekten oder für sicherheitskritische Produkte die UML nach wie vor eine sehr wichtige Rolle spielt.

Aus unserer Sicht ist es notwendig, die Stellung der UML in der Lehre kritisch zu hinterfragen. Wir haben in diesem Beitrag dazu empirische Belege über die Nutzung der UML zusammengetragen. Diese Bestandsaufnahme diente als Grundlage für die Ableitung von Konsequenzen für die Ausgestaltung der Lehre. Naturgemäß kann dies keine abschließende Bewertung sein, sondern soll entsprechend unserer Zielsetzung Impulse zu diesem Thema liefern.

Für weiterführende Arbeiten erscheint neben einer Verbreiterung und Aktualisierung der empirischen Basis auch eine Ausdifferenzierung hinsichtlich verschiedener Studienrichtungen und -niveaus sinnvoll. Es muss beobachtet werden, wie sich die Nutzung der UML in der Praxis weiterentwickelt. Dabei sind Situationen zu betrachtet, in denen Modelle einen Mehrwert leisten. Wer ist Modellersteller, wer -nutzer? Wie umfangreich und präzise müssen die Modelle sein? Müssen aufkommende Programmierparadigmen wie funktionale und deklarative Programmierung berücksichtigt werden, da UML unmittelbar mit OOP zusammenhängt? Wie kann Fachlichkeit stärker in den Fokus genommen werden? Wie kann der identifizierte Kompetenzbedarf in praxisorientierten Lehrkonzepten umgesetzt werden? Wir hoffen, mit diesem Beitrag die Diskussion dazu ein Stück weit angeregt zu haben.

Literaturangaben

Baltes, S., & Diehl, S. (2014). Sketches and diagrams in practice. In S.-C. Cheung, A. Orso, & M.-A. D. Storey (Hrsg.), *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014* (S. 530–541). ACM. https://doi.org/10.1145/2635868.2635891

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Manifesto for Agile Software Development. Abgerufen 30. November 2018, von http://agilemanifesto.org/

Boberić-Krstićev, D., Tešendić, D., Simos, T. E., Psihoyios, G., Tsitouras, C., & Anastassi, Z. (2011). Teaching Object-Oriented Modelling Using UML. In *Numerical Analysis and Applied Mathematics ICNAAM 2011* (S. 810–812). AIP. https://doi.org/10.1063/1.3636856

Bourque, P., Fairley, R. E., & IEEE Computer Society. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3. Aufl.). IEEE Computer Society Press.

Brown, S. (2018). The C4 model for software architecture. Abgerufen 21. Oktober 2018, von https://c4model.com/

Burgueño, L., Vallecillo, A., & Gogolla, M. (2018). Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, *28*(1), 23–41. https://doi.org/10.1080/08993408.2018.1462000

Chaudron, M. R. V., Heijstek, W., & Nugroho, A. (2012). How effective is UML modeling ? *Software & Systems Modeling*, *11*(4), 571–580. https://doi.org/10.1007/s10270-012-0278-4

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game* (0002 Aufl.). Upper Saddle River, NJ: Addison Wesley Pub Co Inc.

Coplien, J. O., & Bjørnvig, G. (2010). *Lean architecture for Agile software development*. Hoboken, N.J.: Wiley.

Davies, I., Green, P., Rosemann, M., Indulska, M., & Gallo, S. (2006). How do practitioners use conceptual modeling in practice? *Data & Knowledge Engineering*, *58*(3), 358–380. https://doi.org/10.1016/j.datak.2005.07.007

Dobing, B., & Parsons, J. (2010). Dimensions of UML Diagram Use: Practitioner Survey and Research Agenda. *Principle Advancements in Database Management Technologies: New Applications and Frameworks*, 271–290. https://doi.org/10.4018/978-1-60566-904-5.ch013

Engels, G., Hausmann, J. H., Lohmann, M., & Sauer, S. (2006). Teaching UML Is Teaching Software Engineering Is Teaching Abstraction. In J.-M. Bruel (Hrsg.), *Satellite events at the MoDELS 2005 conference* (Bd. 3844, S. 306–319). Berlin: Springer. https://doi.org/10.1007/11663430_

Erickson, J., & Siau, K. (2007). *Can UML Be Simplified? Practitioner Use of UML in Separate Domains*.

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software* (1 edition). Boston: Addison-Wesley Professional.

Gesellschaft für Informatik. (2016). *Empfehlungen für Bachelor- und Masterprogramme im Studienfach*

Informatik an Hochschulen (GI-Empfehlungen). Abgerufen von https://gi.de/fileadmin/GI/Hauptseite/Aktuelles/Meldungen/2016/GI-Empfehlungen_Bachelor-Master-Informatik2016.pdf

Gianna Reggio, Maurizio Leotta, Filippo Ricca, & Diego Clerissi. (2013). What are the used UML diagrams? A Preliminary Survey. In *EESS-MOD@MoDELS*.

Glinz, M. (2008). Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. *Informatik-Spektrum*, *31*(5), 425–434. https://doi.org/10.1007/s00287-008-0273-x

Gorschek, T., Tempero, E., & Angelis, L. (2014). On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software*, *95*, 176–193. https://doi.org/10.1016/j.jss.2014.03.082

Jacobson, I. (2009). Taking the temperature of UML. Abgerufen 1. November 2018, von http://blog.ivarjacobson.com/taking-the-temperature-of-uml/

Langer, P., Mayerhofer, T., Wimmer, M., & Kappel, G. (2014). On the usage of UML: initial results of analyzing open UML models. In H.-G. Fill (Hrsg.), *Modellierung 2014*. Bonn: Köllen.

Larman, C., & Vodde, B. (2009). *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Upper Saddle River, NJ: Addison-Wesley.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises*. Upper Saddle River, NJ: Pearson Education.

Liebel, G., Heldal, R., & Steghofer, J.-P. (2016). Impact of the Use of Industrial Modelling Tools on Modelling Education. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)* (S. 18–27). IEEE. https://doi.org/10.1109/CSEET.2016.18

Osman, H., & Chaudron, M. R. V. (2013). UML Usage in Open Source Software Development: A Field Study. Gehalten auf der EESS-MOD@MoDELS 2013.

Petre, M. (2013). UML in Practice. In *Proceedings of the 2013 International Conference on Software Engineering* (S. 722–731). Piscataway, NJ, USA: IEEE Press. Abgerufen von http://dl.acm.org/citation.cfm?id=2486788.2486883

Rupp, C., Queins, S., & SOPHISTen, die. (2012). *UML 2 glasklar: Praxiswissen für die UML-Modellierung* (4. Aufl.). München: Carl Hanser Verlag.

Seiko Akayama, Birgit Demuth, Timothy Lethbridge, Marion Scholz, Perdita Stevens, & Dave R. Stikkolorum. (2013). Tool Use in Software Modelling Education. In *EduSymp@MoDELS*.

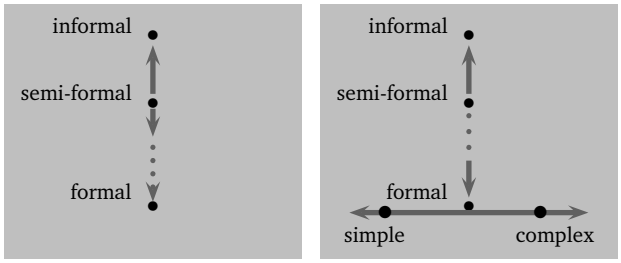
Sommerville, I. (2015). *Software Engineering, Global Edition* (10th edition). Boston: Pearson.

Formale Methoden in der Softwaretechnik-Vorlesung

Bernd Westphal, Albert-Ludwigs-Universität Freiburg
westphal@informatik.uni-freiburg.de

Zusammenfassung

Dieser Artikel stellt die Ergänzung einer Einführung in die Softwaretechnik um Formale Methoden im Bereich Requirements Engineering, Software-Modellierung und Qualitätssicherung vor. Wir beschreiben die Konstruktion der Veranstaltung und berichten empirische Daten und Erfahrungen aus 4 Jahren der Durchführung. Ein Schwerpunkt ist die Beschreibung der verwendeten didaktischen Methoden zur Vermittlung der neuen Inhalte.



(a) (b)

Abbildung 1: Einführung Formaler Methoden.

Abstract

This article presents the extension of an introductory course on software engineering by formal methods in the topic areas requirements engineering, software modelling, and quality assurance. We describe the construction of the course and report empirical data and experience from 4 years of conducting the course. A focus of this article is the description of didactical methods employed for teaching the new content.

1 Einleitung

Im Studienplan des Bachelorstudiums der Informatik der Universität Freiburg ist eine einsemestrige Veranstaltung zur Einführung in die Disziplin der Softwaretechnik vorgesehen. Das Modulhandbuch fordert, wie in Deutschland nicht unüblich, eine Übersicht über die Problembereiche, Konzepte und Techniken der Softwaretechnik, wie sie von den bekannten Lehrbüchern, etwa (Sommerville, 2010; Balzert, 2009; Ludewig u. Lichter, 2013), abgedeckt werden. Im Jahre 2015 stand eine Überarbeitung der Vorlesung ‚Softwaretechnik' an, über deren Resultat wir in diesem Artikel berichten.

Zwei Ziele standen bei der Überarbeitung der Materialien im Fokus. Erstens ein „Zurechtrücken" der Proportionen der verschiedenen Bereiche der Softwaretechnik in den Materialien (die über mehrere Jahre von verschiedenen Veranstaltern verwendet und bearbeitet wurden), und zweitens eine neue Ergänzung der Materialien um eine verständliche und nachvollziehbare Einführung formaler Beschreibungssprachen und Analysetechniken (kurz Formale Methoden), also Beschreibungssprachen mit formal definierter Syntax und Semantik und darauf aufbauenden (Semi-)Entscheidungsprozeduren. Das Ziel der Ergänzung um Formale Methoden ist motiviert durch unsere Erfahrung in zahlreichen Projekten mit Partnern aus der Industrie von kleinen Anbietern sicherheitskritischer Systeme (z.B. (Arenis u. a., 2016)) bis hin zu großen Automobilanbietern und Zulieferern (z.B. (Langenfeld u. a., 2016)). In der Projektarbeit nehmen wir eine klare Nachfrage nach und Anwendung (!) von Formalen Methoden im oben genannten Sinne in verschiedenen Bereichen der Softwaretechnik in den Firmen wahr. Hierbei steht üblicherweise nicht die vollständige und umfassende formale Beschreibung von Anforderungen und Entwurfsentscheidungen bzgl. Struktur und Verhalten im Vordergrund (wie es etwa im Luft- und Raumfahrtbereich teilweise angestrebt wird), sondern eine angemessene Verwendung von Formalen Methoden, um bestimmte, besonders kritische oder risikobehaftete Aspekte von Software zu beschreiben und zu analysieren. Dem entsprechend möchten wir *heute* beginnen, die in vielen Einführungen in die Softwaretechnik fest etablierte (semi-formale) Modellierung von Softwareaspekten in unserer Einführungsveranstaltung um eine formale Sicht zu erweitern und den Studierenden eine Übersicht über Konzepte, Möglichkeiten und Voraussetzungen Formaler Methoden für ihre zukünftigen Berufssituationen zu vermitteln.

Mit diesem Ziel ergibt sich neben der üblichen Herausforderung, aus dem umfangreichen Angebot der Lehrbücher eine ausgewogene Vorlesung zusammenzustellen, die Herausforderung, Lehrmaterialien für den Bereich Formale Methoden auszuwählen bzw. herzustellen. Interessanterweise ist die zweite Her-

ausforderung keine Teilmenge der ersteren. In den etablierten Lehrbüchern (s.o.) und dem vorhandenen Material wird ganz überwiegend ein Ansatz verfolgt, den wir in Abbildung 1(a) visualisieren. Wir sehen die für z.B. die Beschreibung von Anforderungen verfügbaren Mittel auf einer Achse, die von informal (rein natürlichsprachlich), über semi-formal (mit formaler konkreter Syntax, aber ohne formale abstrakte Syntax oder Semantik) hin zu formal (mathematisch präzise definierte Syntax und Semantik) reichen. Die vorgenannten Lehrbücher und Materialien bewegen sich auf dieser Achse zwischen informal und semiformal und geben kurze Ausblicke auf Formale Methoden, indem Beispiele auf einer intuitiven Ebene diskutiert werden. Bei diesem Ansatz sehen wir ein inhärentes Problem, das bereits Lehman & Buth (Lehmann u. a., 2015) beschreiben: „Es darf nicht erwartet werden, dass die Studierenden UML Modelle erstellen können, wenn in der Vorlesung nur die einzelnen Symbole durchgearbeitet werden und dann auf dieser Basis in der Prüfung Analyseaufgaben auf einem gegebenen Diagramm gestellt werden." Streng genommen können wir schon nicht erwarten, daß Studierende in Bloom's Taxonomie echt leichtere Aufgaben, wie etwa *gegebene* Objektdiagramme daraufhin analysieren, zu welchem der (drei) Wahrheitswerte eine *gegebene* OCL-Formel ausgewertet wird, in Übungen so zu bearbeiten, daß gegenüber den Tutor::innen die Korrektheit der Lösung argumentiert (im besten Falle: bewiesen) werden kann bzw. von Seiten der Tutor::innen die Inkorrektheit eines Lösungsvorschlags bewiesen und systematisch erklärt werden kann. Unser Lösungsansatz besteht darin, in der Veranstaltung Endpunkte der Formalitätsachse zu behandeln. Wir diskutieren sowohl informale Ansätze (und im Themenbereich Requirements Engineering verschiedene semi-formale Ansätze) und springen dann zu vollständig definierten Beschreibungssprachen, die für den Zweck der Vorlesung (bis auf eine Ausnahme) auf einen essentiellen Kern reduziert sind (vgl. Abbildung 1(b)). Auf diese Weise geben wir eine konkrete Übersicht über die gesamte Achse. Für eine ausgewählte Beschreibungssprache diskutieren wir mehr als einen essentiellen Kern, um zu demonstrieren, daß es eine orthogonale Achse der Einfachheit formaler Beschreibungssprachen gibt (Einfachheit sowohl bzgl. der Ausdrucksstärke als auch bzgl. des Aufwands, zu einer vollständigen Definition (von abstrakter Syntax und Semantik) zu gelangen).

Für den Bereich der Formalen Methoden in der Softwaretechnik steht mit (Bjørner, 2006a,b,c) ein dreibändiges Lehrbuch zur Verfügung, das versucht, die Softwareentwicklung vollständig aus formaler Sicht zu vermitteln. Wir haben uns aus zwei Gründen dagegen entschieden, diesem Lehrbuch zu folgen. Zunächst scheint uns das Ziel des Materials von Bjørner zu sein, formale Beschreibungssprachen möglichst in vollem Umfang darzustellen, was schlicht den Rahmen unserer Veranstaltung sprengen würde und für unsere Lernziele nicht notwendig ist. Weiterhin ist das Material u.E. so weit von der (heutigen und mutmaßlich morgigen) Lebenswirklichkeit und den etablierten Lehrbüchern entfernt, daß wir annehmen müssen, daß es Studierenden unnötig schwer fallen würde, andere Lehrbücher oder Fortbildungen in der Arbeitssituation zu unserer Veranstaltung in Bezug zu setzen. Unser (zugegeben: hoher) Anspruch an unsere Veranstaltung ist, daß nach der Teilnahme an unserer Veranstaltung keines der etablierten Lehrbücher große konzeptionelle Überraschungen bereit hält, sondern nur von den Studierenden einordnebares, zusätzliches Hintergrundwissen sowie weitere Techniken liefert.

In diesem Artikel beschreiben wir die Konstruktion unserer Vorlesung zur Einführung in die Softwaretechnik im Grundstudium, in der die in etablierten Lehrbüchern zur Softwaretechnik (s.o.) vorgestellten Inhalte um Techniken zur Beschreibung von Anforderungen und Entwurfsaspekten vollständig formal, also mit konkreter Syntax und präziser abstrakter Syntax und Semantik, *ergänzt* werden, und berichten von Erfahrungen aus vier Sommersemestern mit der neu gestalteten Veranstaltung. Schwerpunkt dieses Artikels sind die Voraussetzungen der Veranstaltung (insbesondere die Einbettung in den Studienplan), Konsequenzen der ergänzten Inhalte für Übungen und Klausur, sowie Organisation und didaktische Maßnahmen (vor allem Bereich des Übungsbetriebs), mit denen wir die Vermittlung und Erarbeitung der neuen Inhalte unterstützen. Die neuen Inhalte als solche werden für den Bereich Requirements Engineering in (Westphal, 2018) vorgestellt und sind zudem auf der Homepage der Veranstaltung frei zugänglich.[1]

Der Artikel ist wie folgt strukturiert. In Abschnitt 2 beschreiben wir die Situation der Veranstaltung im Studienplan der Universität Freiburg sowie der Studierenden zu Beginn der Veranstaltung. Abschnitt 3 legt unsere Lernziele dar und beschreibt, wie wir die neuen Inhalte bzgl. Formaler Methoden durch didaktische Maßnahmen im Übungsbetrieb unterstützen. Einen Überblick über die neuen Inhalte, ihre Einbettung in die gesamte Vorlesung und unsere Erfahrungen bzgl. Lernerfolgen gibt Abschnitt 4. Wir schließen den Artikel mit einer Betrachtung von Evaluationsergebnissen und einer Zusammenfassung.

2 Kontext der Veranstaltung und Situation der Studierenden

Die Veranstaltung ‚Softwaretechnik' ist eine Pflichtvorlesung im B. Sc.-Studiengang Informatik der Universität Freiburg mit einem Umfang von 3+1 SWS für Vorlesung und Übung und 6 ECTS-Punkten. Zusätzlich besuchen Studierende des nicht-konsekutiven Masterstudiengangs, der Studiengänge ‚Embedded

[1] Die URL für das Sommersemester 2018 ist: `https://swt.informatik.uni-freiburg.de/teaching/SS2018/swtvl`

Tabelle 1: Selbsteinschätzung der Erfahrung (Minimum · 1., 2., 3. Quartil · Maximum).

	Proj. Mgmt.	Req. Eng.	Programm.	Modellierung	QA
2016 (77 Antw.)	–	0 · 1 · 1 · 3 · 9	1 · 2 · 3 · 5 · 10	0 · 1 · 1 · 2 · 7	0 · 1 · 2 · 3 · 9
2017 (87 Antw.)	0 · 0 · 1 · 3 · 10	0 · 1 · 1 · 4 · 10	0 · 2 · 3 · 5 · 10	0 · 1 · 1 · 3 · 10	0 · 1 · 1 · 4 · 10
2018 (64 Antw.)	0 · 0 · 1 · 3 · 8	0 · 0 · 1 · 2 · 8	1 · 1 · 3 · 5 · 10	0 · 1 · 1 · 3 · 9	0 · 1 · 2 · 4 · 10

Systems Engineering‘ (ESE) und ‚Mikrosystemtechnik‘ sowie des bivalenten Studiengangs Informatik die Veranstaltung. In den vier Jahren der Veranstaltung haben wir zwischen 80 und 150 angemeldete Teilnehmer::innen beobachtet. Unter den Teilnehmer::innen (der Evaluation[2]) studierten zwischen 73 und 96 % Informatik, der zu 100 % fehlende Teil wird stark von ESE-Studierenden dominiert. Außer in 2016 strebten jeweils mehr als knapp zwei Drittel der Teilnehmer::innen einen B. Sc.-Abschluss an.

Die ‚Softwaretechnik‘ ist im 4. Semester vorgesehen, d.h. wir können einen zweiteiligen Programmierkurs und Algorithmen & Datenstrukturen, Technische und Praktische Informatik (Betriebssysteme, Netzwerke, Datenbanken) und die Mathematik-Vorlesungen voraussetzen. Logik und die Einführung in die Theoretische Informatik sind in Freiburg im 3. Semester vorgesehen, liegen also auch zeitlich vor der ‚Softwaretechnik‘. Parallel zur ‚Softwaretechnik‘ findet für die B. Sc.-Studierenden der Informatik ein Softwarepraktikum statt (vgl. Abschnitt 3).

Entsprechend der Position der ‚Softwaretechnik‘ im Curriculum und aus der Erfahrung der Jahre vor 2015 gehen wir einerseits davon aus, daß der Großteil der Teilnehmer::innen vor unserer Veranstaltung bzgl. der meisten Themenbereiche der Softwaretechnik über wenig bis sehr wenig Erfahrung verfügt. Andererseits wissen wir aus persönlichen Gesprächen, daß es im nicht-konsekutiven M. Sc.-Programm durchaus einzelne Studierende gibt, die über mehrjährige Berufserfahrung verfügen.

Seit 2016 überprüfen wir diese Annahme durch eine Aufgabe auf dem ersten Übungsblatt, in der wir die Teilnehmer um eine Selbsteinschätzung der Vorkenntnisse in den Themenbereichen (die wir in der ersten Vorlesung jeweils grob umreißen) Projektmanagement (seit 2017), Requirements Engineering, Programmierung, Modellierung und Qualitätssicherung bitten. Wir schlagen in der Aufgabe die folgende, subjektiv zu interpretierende Skala vor (die Übungsblätter sind in Englisch verfasst):

0: I have no experience in that activity whatsoever. I have not taken any related subjects during my studies.

1: I have only performed the activity in the context of a lecture or programming course.

10: I have performed the activity in a project with a large user base (100+ users), a large work volume (36+ person-months) or a specific commercial purpose. I have been responsible for the planning and execution of the activity in a software development project within defined resource and time constraints.

[2] Zwischen 26 und 40 Angaben; leider haben wir keinen direkten Zugriff auf Studiengang oder angestrebten Abschluss.

Tabelle 1 zeigt die bisherigen Ergebnisse, die unsere Annahmen durchweg bestätigen. Etwas überraschend finden wir, daß sich das obere Quartil der Studierenden im Bereich Programmierung regelmäßig als relativ erfahren einschätzte, sich im Bereich Qualitätssicherung (der die Aktivität des Testens einschließt, von der man erwarten sollte, daß diese mit ernsthafter Programmierung einhergeht) als eine oder zwei Stufen geringer erfahren einordnet.

In der zweiten Vorlesung zeigen und diskutieren wir die Ergebnisse des jeweiligen Jahres. Mit der Präsentation der Ergebnisse verbinden wir die folgenden Botschaften: Erstens, daß wir, d.h. die Veranstalter und die Studierenden, es (auch dieses Jahr wieder) mit einem bzgl. Vorerfahrung eher heterogenen Auditorium zu tun haben. Zweitens, daß die Vorlesung explizit für Studierende angelegt ist, die sich zwischen den Werten 0 und 1 einordnen (und daß diese Studierenden in der klaren Mehrheit sind, man also nicht „allein“ ist, wenn man sich dort eingeordnet hat). Drittens, daß für die Studierenden, die sich bei Werten von 5 und höher einordnen, möglicherweise keine Neuigkeiten vermittelt werden, daß diese Studierenden jedoch explizit eingeladen sind, in Form von Fragen oder Kommentaren ihre Vorerfahrung in den Vorlesungs- und Übungsterminen einzubringen. In der Diskussion der Ergebnisse stellen wir heraus, daß die Inhalte der Vorlesung die Studierenden darauf vorbereiten möchten, in Softwarentwicklungsprojekten höherer Stufen mitzuarbeiten, also Projekte die einen größeren Umfang und eine größere Nutzerbasis haben als jedes Projekt, das man im Studium in Freiburg erlebt, sowie üblicherweise einen wirtschaftlichen Aspekt haben.

3 Ziele und Konstruktion der Veranstaltung

Ein Lernziel unserer Veranstaltung ist zunächst, wie in der Einleitung beschrieben, eine Übersicht über die verschiedenen Themenbereiche der Softwaretechnik entsprechend der etablierten Lehrbücher zu geben. Unser Schwerpunkt ist hierbei die Vermittlung der verschiedenen Probleme, die sich daraus ergeben, daß mehrere Menschen über einen längeren Zeitraum zusammen nichttriviale Softwaresysteme entwickeln, sowie bekannte Lösungsansätze und deren Vor- und Nachteile vorzustellen.

Der neue Aspekt unserer Veranstaltung ist die vollständig formale Definition von Beschreibungssprachen und Analysetechniken an den entsprechenden Stellen in der Vorlesung (vgl. Tabelle 2). Die Lernziele sind hier das sichere Verstehen und Anwenden der eingeführten Formalen Methoden, sowie die Interpretation von Analyseergebnissen. Die entsprechenden Inhalte und die Einbettung in die Vorlesung werden in Abschnitt 4 beschrieben.

Wir beschreiben die Ziele unserer Veranstaltung in der ersten Vorlesung, um Missverständnisse (und Enttäuschungen) zu vermeiden. In einer Aufgabe des ersten Übungsblattes bitten wir die Studierenden, ausgehend von der ersten Vorlesung ihre Erwartungen an die Veranstaltung in eigenen Worten zu beschreiben. In der zweiten Vorlesung stellen wir eine Auswahl der Antworten vor, und gehen jeweils darauf ein, inwiefern die Veranstaltung diese Erwartungen erfüllen kann. Ein über die Jahre wiederkehrender Punkt, ist der Wunsch, gutes Design zu lernen. Diese Erwartungen liefern einen guten Anlass, erneut auszuführen, daß ein Ziel der Vorlesung ist, fundamentale Designprobleme zu verstehen und Designentscheidungen präzise zu beschreiben und auf Aspekte guten Designs hin zu analysieren, wir jedoch keine konkreten Verfahren zur Erstellung guter Designs für spezifische Anwendungsfelder diskutieren. Wir plausibilisieren diese Zielsetzung durch eine grobe Beschreibung des weiten Spektrums von Software: von Onlineshops, über Standardanwendungen (Textverarbeitung, etc.), kommerzielle Spiele, Betriebswirtschaftliche Software, bis hin zu sicherheitskritischen, eingebetteten Systemen. Jeder dieser Bereiche hat eigene, bewährte Vorstellungen von gutem Design, die untereinander nicht austauschbar sein müssen.

3.1 Klausur

Ähnlich wie (Lehmann u. a., 2015), haben wir sehr früh in der Überarbeitung der Veranstaltung mit der Vorbereitung der Klausur begonnen. Laut Modulhandbuch ist eine schriftliche Prüfung vorgesehen und somit war klar, daß die Vorlesung notwendig schriftlich prüfbare Inhalte in hinreichendem Umfang und Tiefe präsentieren muß. Um das Risiko zu mindern, zum zentral festgelegten Klausurtermin keine geeignete Klausur stellen zu können, haben wir, ausgehend von einem ersten Entwurf des Zuschnitts der Vorlesung, Aufgabenskizzen und eine grobe Festlegung der Proportionen für die Themenbereiche erstellt. Vorlesungsinhalte, Klausuraufgaben und Übungsinhalte wurden dann im Sinne eines guten *Constructive Alignments* (Biggs u. Tang, 2011) aufeinander abgestimmt. Zum Ende der Vorlesungszeit 2015 stand für die Studierenden zur Klausurvorbereitung eine Beispielklausur zur Verfügung, seit 2016 ist die Beispielklausur mit dem ersten Vorlesungstag auf der Lernplattform verfügbar.

Bei der Punkteverteilung der Klausur orientieren wir uns grob an den Stufen der (überarbeiteten) Taxonomie der Lernziele (Bloom, 1956; Anderson u. a., 2001). Circa 50 % der Punkte entfallen auf „leichte“ Aufgaben (Vokabular, einfache Verfahren verstehen und wie in den Übungen (jedoch auf andere Probleme) anwenden), ca. 33 % der Punkte entfallen auf „mittlere“ Aufgaben (ähnlich wie Übungsaufgaben, jedoch Analyse bzw. Transfer notwendig) und ca. 17 % der Punkte sind für „schwere“ Aufgaben (nicht in den Übungen besprochen, neue Sichten auf diskutierte Konzepte, offenere kreative Aufgaben) vorgesehen. Unsere Absicht ist, Kompetenzen über alle kognitiven Kategorien hinweg zu prüfen; in der o.g. Punkteverteilung reicht es z.B. zum Bestehen nicht aus, alle „schweren“ Aufgaben (ggf. intuitiv oder zufällig) zu lösen, sondern es sind Punkte aus den Bereichen Verstehen und Anwenden zum Bestehen notwendig.

3.2 Vorlesungen

Teil unserer Veranstaltung ‚Softwaretechnik‘ ist eine klassische Vorlesung im Umfang von 3 SWS. Um nicht an einem Tag der Woche auf eine Stunde Tutorium umschalten zu müssen, interpretieren wir das 3+1-Format als drei Vorlesungen (à 90 Minuten) und ein Tutorium (à 90 Minuten) in zwei Wochen.

Über die vorhandene Infrastruktur zeichnen wir das gesprochene Wort und den Bildschirminhalt auf und stellen die Aufzeichnungen zeitnah zur Verfügung. Auf diese Weise konnten wir die Vorlesungen ohne Skript bestreiten, da das gesprochene Wort jederzeit zum Nachhören verfügbar ist. Weiterhin können die Teilnehmer::innen, denen unser Vortrag zu langsam oder zu ausführlich ist, die Geschwindigkeit beim Anhören der Aufzeichnung selbst bestimmen. Entsprechend der Selbsteinschätzung der Teilnehmer::innen (vgl. Abschnitt 1) sprechen wir in der Vorlesung lieber zu langsam als zu schnell und betrachten neue Inhalte ggf. aus mehreren Perspektiven, was für einige Teilnehmer::innen zu langsam sein kann. In der Evaluation geben, über die Jahre 2016 bis 2018 aggregiert,[3] von 99 Rückmeldungen knapp 25 % an, sich die Inhalte hauptsächlich durch Anwesenheit anzueignen, und 18 % bzw. 33 % geben ‚teils/teils‘ bzw. ‚hauptsächlich über die Aufzeichnungen‘ an. Entsprechend haben wir eher geringe Besucherzahlen in der eigentlichen Vorlesung, dafür jedoch Besucher::innen, die die Präsenz für Fragen oder Kommentare nutzen möchten, denen wir, so gut es die Zeit erlaubt, Raum geben.

Das Lernziel, eine Übersicht entsprechend der etablierten Lehrbücher zu geben, hat für uns zwei Teilaspekte. Einerseits den Aspekt der Vermittlung von Techniken und Verfahren und andererseits den Aspekt „der Mensch steht im Mittelpunkt“ (Ludewig u. Lichter, 2013). Der erste Aspekt ist relativ gut prüf-

[3] Im Jahr 2015 wurde in der Evaluation zum Vortragsbesuch eine andere, nicht gut vergleichbare Frage gestellt.

bar (und kann als Vorbereitung auf die Klausur gesehen werden). Zum zweiten Aspekt ist unser Ziel, das (gesprochene und schriftliche) Kommunizieren von Softwaretechnik-Problemen und Lösungsideen und die Analyse und Bewertung von Lösungsideen einzuüben. Dieser zweite Aspekt ist ungleich schwerer prüfbar (und kann als Vorbereitung auf „das echte Leben“ gesehen werden) und findet in unserer Veranstaltung schwerpunktmäßig im Übungsbetrieb statt (siehe folgender Abschnitt).

3.3 Übungsaufgaben und Tutorate

Die Vorlesung wird von Übungen begleitet. Vor der ersten Vorlesung jedes 3er Blocks steht ein Übungsblatt zur Verfügung, das direkt nach dieser Vorlesung teilweise und nach der zweiten Vorlesung des Blocks vollständig bearbeitet werden kann, sodaß für jede Aufgabe mindestens eine Woche zur Bearbeitung zur Verfügung steht. Studierende bearbeiten die Aufgaben in Teams aus 2 bis 3 Personen und können ihre Bearbeitungen bis zur Minute vor Beginn des Tutoriums auf der Lernplattform einreichen.

In den Tutorien legen wir Wert auf Interaktion (i.S.v. (Krusche u. a., 2017)). Das heißt, es sind nicht die Tutor::innen, die eine Beispiellösung vorstellen, die „passiv konsumiert“ werden kann („man trifft sich nicht mehr nur, um ‚gemeinsam zuzuhören‘“ (Siegeris, 2017)),[4] sondern der Modus der Tutorien ist „wir erarbeiten eine gute Lösung zusammen“ bzw. „wir analysieren und bewerten Lösungsvorschläge und diskutieren weiterführende Fragen“. Die Tutor::innen sehen wir im Tutorium vor allem als Moderator::in, nicht als Vortragende::r. Hierbei sollen die Lösungswege und -überlegungen transparent werden. Zu diesem Zweck sind die Tutor::innen angehalten, für die technischen Aufgaben (im Sinne des Retrieval-Based-Learning) die Teilnehmer::innen zunächst nach den relevanten Definitionen zu fragen. Bei kleinen Aufgaben schreiben die Tutor::innen Lösungsvorschläge aus dem Auditorium auf den Bildschirm, bei größeren Aufgaben bringen die Tutor::innen bemerkenswerte Lösungen mit ins Tutorium und stellen sie zur Diskussion. Die bemerkenswert guten oder eigenartigen Lösungen werden aus sogenannten *Early Submissions* ausgewählt. Wir bieten denjenigen Teams, die 24 Stunden vor dem Tutorium Lösungen einreichen, einen 10 %-Bonus auf die erzielten Zulassungspunkte. Beispiele für weiterführende Fragen ergeben sich z.B. im Bereich des Requirements Engineering wie folgt. Eine technische Aufgabe kann darin bestehen, eine gegebene Formalisierung einer Anforderung auf (formale) Vollständigkeit zu untersuchen. Ausgehend von der (technischen) Lösung ergibt sich die Frage, wie das Resultat in der Rolle Requirements Engineer zu interpretieren ist und welche Konsequenzen für die Kommunikation mit der Kundenseite ggf. zu ziehen sind. Weiterführende technische Fragen können konstruiert werden, indem man die gezeigte Problemstellung leicht modifiziert und eine erneute Analyse erfragt. Bei der Diskussion von Lösungen und weiterführenden Fragen achten die Tutor::innen auf präzise Sprache, insbesondere die korrekte Verwendung von Softwaretechnik-Fachsprache.

Für die schriftliche Darstellung in den Einreichungen fordern wir regelmäßig die Form „Problem in eigenen Worten darstellen – eigenen Lösungsvorschlag formulieren – begründen, argumentieren, im besten Falle: beweisen, daß der Lösungsvorschlag das definierte Problem löst“ und halten die Tutor::innen dazu an, in Kommentaren zu den Einreichungen kontinuierlich diese Form nachzufragen.[5] Entsprechend sind die (ausgewogen vorkommenden) offenen Modellierungsaufgaben absichtlich unpräzise formuliert. Hierbei ist unserer Erfahrung nach der angemessene Grad an Unschärfe in der Aufgabenstellung im B. Sc.-Programm signifikant geringer als im M. Sc.-Programm. Diese absichtliche Unschärfe und die didaktische Absicht kommunizieren wir ganz explizit: wir gehen (stark) davon aus, daß die allerwenigsten Absolventen in ihrer beruflichen Laufbahn präzise Aufgabenstellungen bekommen werden — und unter dieser Annahme können wir ruhig im 4. Semester langsam mit Aufgaben dieser Art beginnen. Im ersten Jahr der Veranstaltung gab es in der Evaluation vereinzelt Wünsche nach ganz klaren Aufgaben. Unsere Hypothese ist, daß in den meisten vorherigen Veranstaltungen präzise Aufgabenstellungen vorherrschen (und den Lernzielen angemessen sind, vgl. auch Abschnitt 4.2) und die ‚Softwaretechnik‘ sich in diesem Aspekt für die Studierenden ungewohnt anfühlt.

Am Tag vor den Tutorien führen wir jeweils ein Tutorium für die Tutor::innen durch, vor dem die Tutor::innen das Übungsblatt selbst skizzenhaft lösen. In diesen *Tutors' Tutorials* gehen wir (in der Art der Tutorien für die Studierenden) die Aufgaben durch, diskutieren weiterführende Fragen, legen die Lernziele der jeweiligen Aufgaben und Fragen dar und geben Tips zur Moderation. Der Ablauf der Tutorien steht danach inklusive Vorschlägen für weiterführende Fragen und Lernziele als eine Art „Drehbuch“ zur Verfügung. In der ersten Durchführung hatten wir leichte Sorge, daß sich die Tutor::innen in ihrer gestalterischen Freiheit beschränkt fühlen könnten. Hier waren die Rückmeldungen bisher durchweg positiv: ganz im Gegenteil würden die *Tutors' Tutorials* Sicherheit (z.B. im Zeitmanagement) geben und kognitive Kapazitäten für tiefer gehende Fragen und Kommentare von Seiten der Studierenden freihalten. Weiterhin erreichen wir durch diesen Ansatz eine überwiegend gleichbleibende Qualität der Tutorien, unabhängig von dem/der konkreten Tutor::in.

[4] Auch wenn einige Studierende sich dies lt. Evaluation wünschen würden.

[5] Nicht zuletzt im Interesse der Studierenden: Wir gehen davon aus, daß gut präsentierte Lösungsvorschläge bei der Klausurvorbereitung zumindest nicht hinderlich sind.

Tabelle 2: Kursinhalte und Struktur.

Vorlesungen	Themenbereich	Inhaltsübersicht
1	Einleitung	Software, Engineering, Software Engineering; Erfolgreiche Softwareentwicklung, Empirische Daten zum Erfolg von Softwareprojekten; Aufbau des Kurses, Bezug zu anderen Lehrveranstaltungen, Organisatorisches.
2 – 5	Projektmanagement	Softwaremetriken, Skalen; Kostenschätzung (Experten- und algorithmische Schätzung); Projekt, Prozess, Prozess Modellierung; Prozedurmodelle; Prozessmodelle (Agil, V-Modell (*semi-formal*)).
6 – 10	Requirements Engineering	Vokabular: Anforderungen, Anforderungsanalyse; Anforderungen im Entwicklungsprozess; Eigenschaften von Anforderungsspezifikationen (Vollständigkeit, Konsistenz, . . .), Arten von Anforderungen, Analysetechniken; Dictionary; Spezifikationssprachen für Anforderungen: natürlichsprachliche Pattern, Entscheidungstabellen (*formal*), Use Cases und -Diagramme (*semi-formal*), LSCs (*formal*).
11 – 14	Design & Architektur	Modell; Sichten; Strukturmodellierung für Software (Klassen- und Objektdiagramme (*formal*), Proto-OCL (*formal*)); Designprinzipien; Design Patterns; Verhaltensmodellierung für Software (Communicating Finite Automata (*formal*), Query Language (*formal*)); ein Ausblick zu UML State-Machines.
15 – 18	Qualitätssicherung	Testfall, Testausführung, falsch/richtig positive/negative Ergebnisse (*formal*); Grenzen des Softwaretestens; Glas-Box-Testen (Anweisungs-, Verzeigungs-, Term-Überdeckung); Modell-basiertes Testen, Runtime-Verifikation; Programmverifikation (*formal*), Code Review.

In der auf das Tutorium folgenden Woche geben die Tutor::innen individuelle (für die Klausurzulassung relevante) Bewertungen und Kommentare zu den Einreichungen. Hierbei werden die Einreichungen auf zwei verschiedenen Skalen bewertet. Die für die Klausurzulassung relevante *good-will*-Skala bewertet „sinnvolle Bearbeitung“ relativ zum Wissen der Studierenden *vor* dem Tutorium. Hier interpretieren die Tutor::innen unklare Formulierungen im Zweifel zugunsten der Studierenden. Um einen Eindruck von der Bewertung in der Klausur zu geben, bewerten wir außerdem jede Aufgabe auf der *evil*-Skala, die sich am Punkteschema der Klausur und am Wissen *nach* dem Tutorium orientiert, insbesondere darüber, wie bestimmte Aufgabenstellungen im Kontext der Veranstaltung im Zweifel zu interpretieren sind. Hier werden unklare Formulierungen oder Fehler in der Syntax zuungunsten der Studierenden gewertet.

4 Formale Methoden in der Softwaretechnik-Vorlesung

Für die Darstellung von Vokabular, Problembereichen und nicht formalen Inhalten folgt unsere Vorlesung im Wesentlichen dem Lehrbuch (Ludewig u. Lichter, 2013). Wir schätzen an diesem Lehrbuch die explizit adressierte Sicht, „dass sich Software-Engineering vor allem mit den Menschen befasst, die Software in Auftrag geben, entwickeln und ändern oder benutzen“ und das u.E. sehr gelungene „[B]emühen [...] um eine rationale Wertung“ und darum, „alle Quellen [...] vollständig anzugeben“.

Im folgenden beschreiben wir unsere Erweiterung der Inhalte um *formale* Modelle in der Softwareentwicklung, die sich unserer Wahrnehmung nach nahtlos an die Diskussion der Begriffe der Modellierung und die Bedeutung von Modellen in der Softwareentwicklung bereits im ersten Kapitel von (Ludewig u. Lichter, 2013) anfügt. In der Diskussion der (gegenüber (Ludewig u. Lichter, 2013)) neuen Inhalte bemühen wir uns, dem evidenzbasierten Stil des Lehrbuchs zu folgen, also rationale Wertungen zu ermöglichen und soweit möglich die (ursprünglichen) Quellen anzugeben. Entsprechend zeigen wir anhand von Beispielen, was Formale Methoden konkret leisten können, was ihre Fürsprecher::innen versprechen und welche Kritik vorgebracht wird, sodaß die Studierenden Vor- und Nachteile verantwortlich und kontextabhängig abwägen können.

Tabelle 2 gibt einen Überblick über die Inhalte der Vorlesung. Wir beginnen mit dem Themenbereich Projektmanagement, da die Inhalte dieses Bereichs für die Teilnehmer::innen des parallel im 4. Semester stattfindenden Software-Praktikums (SoPra) für B. Sc.-Studierende nützlich sein können. Es gibt von Seiten der Teilnehmer::innen durchaus den Wunsch nach einer Abstimmung zwischen dem SoPra und der ‚Softwaretechnik‘, der aus verschiedenen Gründen nicht praktikabel (und, da das SoPra eine in sich abgeschlossene Veranstaltung mit eigenen Vorlesungen ist, auch nicht notwendig) ist. Ein Grund ist, daß im SoPra bereits in der dritten Woche die Spezifikation und ein Architekturentwurf abzugeben sind. Ein weiterer Grund ist, daß in jedem Jahr über 30 % der ‚Softwaretechnik‘-Teilnehmer::innen das SoPra nicht in ihrem Studienplan haben und Versuche einer engeren Abstimmung bei diesen Teilnehmer::innen zu signifikanten Verwirrungen geführt haben. Als Kompromiss haben einige Übungsaufgaben einen Bezug

zum SoPra, der sich den SoPra-Teilnehmer::innen erschließt, und für die anderen Studierenden einfach „ein Beispielprojekt“ ist.

In der Präsentation und Diskussion der nicht-formalen Inhalte folgen wir (Ludewig u. Lichter, 2013), wobei wir im Zweifel direkt auf die ursprünglichen Quellen zurückgreifen, z.B. die Normtexte IEEE 601.12 und ISO 24765 für Vokabular und IEEE 830 zur Struktur von Anforderungsdokumenten. In den folgenden Abschnitten geben wir einen Überblick über die von uns ergänzten formalen und semi-formalen Inhalte.

Aufgrund unserer in der Einleitung ausgeführten Beobachtung eines Mangels an geeigneten Lehrbüchern, haben wir die formalen Inhalte selbst ausgewählt bzw. konstruiert. Der Überarbeitung der Veranstaltung lag die Hypothese zugrunde, daß es durch eine angemessene Reduktion des Sprachumfangs möglich ist z.B. die formale Spezifikationssprache für Strukturaspekte OCL vollständig definiert einzuführen und die Idee Formaler Methoden und Analysen zu vermitteln, und gleichzeitig der Vermittlung der etablierten nicht-formalen Inhalte einen angemessenen Raum zu geben. Bei der Auswahl der Teilsprachen von z.B. OCL stand das Ziel im Vordergrund, die formalen Sprachen nicht artifiziell zu vereinfachen, sondern echte Teilsprachen auszuwählen, die in Spezialvorlesungen konservativ erweitert werden. Wir konnten hierbei auf eine langjährige Erfahrung aus Forschungs- und Projektarbeit sowie aus unseren Spezialvorlesungen ‚Software, Design, Modelling, and Analysis in UML‘ (UML) und ‚Real-Time Systems‘ im M. Sc. Informatik zurückgreifen, in denen für die Modellierungssprachen jeweils eine vollständige konkrete und abstrakte Syntax sowie eine formale Semantik eingeführt wird. In der UML-Vorlesung sind dies Klassen- und Objektdiagramme, OCL, State-Machines und Sequenzdiagramme.

Bei der Auswahl der Beispiele in der Vorlesung und der Konstruktion der Übungsaufgaben greifen wir soweit möglich auf reale, industrielle Softwareprojekte (bestenfalls aus unserer eigenen Erfahrung) zurück, da so sichergestellt ist, daß wir Nachfragen in beliebiger Detailtiefe zufriedenstellend beantworten können.[6] Dabei verwendet die Vorlesung für die verschiedenen Formalen Methoden verschiedene Beispiele an denen sich die spezifischen Stärken zeigen. Wir möchten (und können redlicherweise) nicht den Eindruck erwecken, daß der Stand der Technik der Formalen Methoden ist, ein Softwaresystem vollständig und durchgängig konsistent formal zu beschreiben. Formale Methoden sind dann effektiv, wenn geeignete Formalismen zur Beschreibung und Analyse spezifischer, besonders kritischer oder komplexer Aspekte von Software mit Bedacht eingesetzt werden.

[6] Dies ist bei artifiziell konstruierten Beispielen ungleich schwerer herzustellen und kann, wenn es nicht gelingt, unnötige Unzufriedenheit auf Seiten der Studierenden hervorrufen.

Wir nutzen bei der Auswahl der Inhalte die relative späte Lage der Veranstaltung im Studienplan aus (vgl. Abschnitt 2). So bauen wir auf ‚Grundlagen der Theoretischen Informatik‘, die Mathematik-Vorlesungen und ‚Rechnerarchitektur‘ auf, in denen die Studierenden bereits mit verschiedenen Formalen Methoden konfrontiert wurden (wenn auch vielleicht nicht unter diesem Namen). Unsere Veranstaltung zeigt gewissermaßen, wie ein großer Teil des bisher Gelernten in der Softwarekonstruktion angewandt werden kann.

4.1 Themenbereich Softwareprojektmanagement

Im Themenbereich Softwareprojektmanagement überwiegen nicht-formale Inhalte (vgl. Tabelle 2). Ein Schwerpunkt sind Softwaremetriken als ein Aspekt der ingenieurmäßigen, objektivierten Softwareentwicklung (inklusive des Bewusstseins für Pseudometriken (vgl. (Ludewig u. Lichter, 2013))). Ein weiterer Schwerpunkt ist die Prozessmodellierung. Hier führen wir zunächst eine semi-formale[7] graphische Beschreibungssprache für Prozesse ein, bestehend aus Rollen, Artefakten, Aktivitäten und den entsprechenden Relationen. Wie zeigen, wie ein konkreter Ablauf (oder Prozess) einer Softwareentwicklung modelliert werden kann (deskriptiv, Abbild), um dann zu zeigen, wie ein graphisches Prozessmodell präskriptiv (Vorbild) verwendet werden kann, um Prozessabläufe vorzugeben. Wir verwenden unsere graphische Beschreibungssprache, um konkrete Prozedur- und Prozessmodelle (Wasserfall, V-Modell, XP, Scrum) vorzustellen bzw. setzen die Notation z.B. der V-Modell-Beschreibung zu unserer Teilsprache in Beziehung.

In den Übungsaufgaben ist anhand von vorgegebenen Prozessbausteinen ein kleines Entwicklungsprojekt zu planen inklusive Besetzung von Rollen mit Personen. Um die Tutor::innen insbesondere auf Fragen und Diskussionen zur Verwendung und zum Nutzen von Prozessmodellen vorzubereiten, haben wir entsprechend des Prinzips „lehren, was wir praktizieren“ in 2018 den Prozess des Übungsbetrieb vollständig modelliert. Die Erläuterung des Prozesses ist aufwendig, dafür weiß über das gesamte Semester jede::r Tutor::in, wer wann welches Artefakt wie zu bearbeiten hat und wir können unsere ganze Aufmerksamkeit den eingereichten Lösungsvorschlägen widmen.

4.2 Themenbereich Requirements Engineering

Der Themenbereich Requirements Engineering (RE) ist in der Veranstaltung etwas stärker gewichtet als die weiteren Themenbereiche (vgl. Tabelle 2). Dies ist vor allem drei Überlegungen geschuldet. Erstens stimmen wir zu (etwa (Sedelmaier u. Landes, 2017)),

[7] d.h. präzise konkrete Syntax, informelle Beschreibung der Bedeutung.

daß dem Themenbereich RE in der Softwareentwicklung eine besondere Relevanz zukommt. Die Relevanz wird dabei in den späteren Themenbereichen sehr natürlich aufgegriffen: Der Begriff der Korrektheit von Softwareentwürfen (insbesondere bzgl. des Verhaltens) und von Software ist *relativ* zu einer Anforderung. Etwa ein Algorithmus kann nur als korrekt (bzgl. einer Anforderung) bewiesen werden, wenn es eine formale Beschreibung der Anforderung gibt. Zweitens beobachten wir, daß die Vermittlung von Problemen und Techniken des RE für Studierende mit wenig oder keiner Vorerfahrung eine besondere Herausforderung darstellt (vgl. Selbsteinschätzung der Studierenden, Abschnitt 2). Drittens führen wir in unserer Veranstaltung im Themenbereich RE zum ersten Mal Formale Methoden ein.

Wir sprechen bei unseren Diskussionen i.d.R. über die u.E. besonders herausfordernde Situation eines Softwareentwicklungsvertrages zwischen getrennten Kunden- und Entwicklungsunternehmen. Die Softwareentwicklung in Start-Ups, innerhalb einer Abteilung oder zwischen Abteilungen desselben Unternehmens hat ggf. andere Rahmenbedingungen.

Bei der Vermittlung von Problemen und Techniken des RE arbeiten wir mit zwei (bisher leider nicht empirisch belegten) Hypothesen zu Ursachen von Schwierigkeiten. Eine Ursache sehen wir darin, daß der weitaus überwiegende Teil der Übungsaufgaben, die die Studierenden bis zum 4. Semester bearbeiten, von Fachleuten in Fachsprache *präzise* formuliert sind (nicht zuletzt, um die Korrektur zu vereinfachen). Entsprechend stellen wir Übungsaufgaben zu unserer Veranstaltung absichtlich unpräzise und halten dazu an, in den Einreichungen zunächst die Aufgabenstellung in eigenen Worten wiederzugeben (vgl. Abschnitt 3.3). Als zweite Ursache vermuten wir, daß der Umgang mit Sprache im Alltag von dem in der RE-Arbeit verschieden ist. Im Alltag sind Menschen üblicherweise gern bereit, einen Satz wie:

> Spielfiguren müssen indirekt gesteuert werden.

direkt als „klar“ zu akzeptieren und keinen Bedarf für Nachfragen zu sehen (fehlende Information wird plausibel aufgefüllt; man sieht nur Information). In der RE-Arbeit geht es unseres Erachtens beim Blick auf den obigen Satz vor allem darum, zu sehen, welche Information der Satz *nicht enthält*.

Um diesen Sichtwechsel anzuregen, führen wir eine Übung durch, in der die Studierenden eine textuell gegebene Anforderung analysieren. Diese Übung wird in verschiedenen Ausprägungen im RE-Unterricht eingesetzt, etwa mit Paaren von studentischen Teams, die jeweils die Kunden- bzw. Entwicklerrolle übernehmen oder mit nicht technisch vorgebildeten (echten) Kunden. Wir verwenden eine Variante mit einem technisch vorgebildeten Kunden, da wir in den anderen Szenarien die Gefahr sehen, daß die Parteien aneinander vorbeireden und es aufwendig ist, eine für alle Beteiligten gute Beispiellösung zu erarbeiten. Unser Kunde ist ein Organisator des SoPra, unsere Anforderung war in 2018 der oben genannte Satz. Die Aufgabe besteht darin, durch Kommunikation mit dem Kunden ein Begriffslexikon zu erarbeiten und die nicht im Satz enthaltenen Informationen „herauszukitzeln“. Um die Aufgabe skalierbar zu gestalten, führen zunächst die Organisatoren unserer Veranstaltung eine Anforderungsanalyse durch und notieren die Findungen in einem Wiki. Die Studierenden kommunizieren mit dem (für sie anonymen) Kunden per Mail, vermittelt durch die Tutor::innen. Mit Hilfe des Wiki können die Tutor::innen die meisten Fragen ihrer Tutanden schon im Stile eines Kunden beantworten, noch unklare Aspekte werden an den SoPra-Organisator weitergeleitet und im Wiki nachgeführt; weiterhin sind die Tutor::innen angehalten, die Kommunikation in der Art eines Coaches zu beobachten und Tips für weitere Fragen zu geben. Für das Tutorium bereiten wir eine Liste von bekannten Beispielspielen vor, die die Anforderung lt. Kunde erfüllen bzw. nicht erfüllen. Im Tutorium sollen die Teilnehmer::innen durch Handzeichen pro Beispiel anzeigen, ob nach ihrer konkreten Anforderungsanalyse das Beispiel vom Kunden akzeptiert oder abgelehnt wird bzw. ob sie sich nicht sicher sind. In 4 Jahren Durchführung dieser Übung hat noch nicht ein Team alle Beispiele korrekt klassifiziert; dies wird durch das Verfahren mit den Handzeichen im Tutorium unmittelbar für die Studierenden erfahrbar.

In der Einführung der Problemstellung der Anforderungsanalyse und nicht-formalen Beschreibung von Anforderungen folgen wir (Rupp u. a., 2014). Wir beginnen die Diskussion von Beschreibungssprachen am nicht-formalen Ende der in Abbildung 1(b) dargestellten Achse mit der natürlichsprachlichen Beschreibung und Sprach-Patterns (Rupp u. a., 2014) und diskutieren Eigenschaften guter Anforderungsdokumente wie Vollständigkeit. Die Präsentation wechselt dann ans formale Ende der Achse und dort zu einem möglichst einfachen Formalismus (vgl. Abschnitt 4.2.1), um dann im semi-formalen Bereich z.B. User Stories und Use Cases einzuführen. Ein komplexer Formalismus (vgl. Abschnitt 4.2.2) und ein Rückblick schließen den Themenbereich ab.

4.2.1 Entscheidungstabellen

Als Einstieg in die formale Beschreibung von Anforderungen haben wir als eine möglichst einfache Sprache die in vielen Lehrbüchern semi-formal eingeführten Entscheidungstabellen (ET) ausgewählt. ET sind eine der einfachsten Beschreibungssprachen, die mit formaler Semantik und Analysen versehen werden können, sind jedoch weder *trivial* noch eine artifizielle Lehrsprache. Die schlichten Sprachmittel von ET reichen vollständig aus, um Anforderungen zu beschreiben, die in textueller Form sehr schnell nicht mehr überblickbar, wartbar oder analysierbar sind.

T: decision table		r_1	$\cdots$	r_n
c_1	description of condition c_1	$v_{1,1}$	$\cdots$	$v_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_m	description of condition c_m	$v_{m,1}$	$\cdots$	$v_{m,n}$
a_1	description of action a_1	$w_{1,1}$	$\cdots$	$w_{1,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_k	description of action a_k	$w_{k,1}$	$\cdots$	$w_{k,n}$

Abbildung 2: Konkrete Syntax von ET.

Formal ist eine ET T über disjunkte Mengen von *Bedingungen* C und *Aktionen* A eine $n \times m$-Matrix (siehe Abbildung 2) wobei $c_1, \ldots, c_m \in C$, $a_1, \ldots, a_k \in A$, $v_{1,1}, \ldots, v_{m,n} \in \{-, \times, *\}$, und $w_{1,1}, \ldots, w_{k,n} \in \{-, \times\}$. Eine wohlgeformte ET hat einen Namen. Spalten $(v_{1,i}, \ldots, v_{m,i}, w_{1,i}, \ldots, w_{k,i})$, $1 \leq i \leq n$, werden *Regeln* genannt und haben einen eindeutigen Namen (hier: $r_1, \ldots, r_n$). Die Vektoren $(v_{1,i}, \ldots, v_{m,i})$ und $(w_{1,i}, \ldots, w_{k,i})$ heißen *Prämisse* und *Effekt* von Regel r_i. Die Semantik einer ET T ist durch die Funktion $\mathcal{F}$ gegeben, die jeder Regel r in T eine Formel der propositionalen Logik über C und A wie folgt zuordnet. Seien $(v_1, \ldots, v_m)$ und $(w_1, \ldots, w_k)$ Prämisse und Effekt von r. Dann ist

$$\mathcal{F}(r) := \underbrace{\textstyle\bigwedge_{1 \leq i \leq m} F(v_i, c_i)}_{=:\mathcal{F}_{pre}(r)} \wedge \underbrace{\textstyle\bigwedge_{1 \leq j \leq k} F(w_j, a_j)}_{=:\mathcal{F}_{eff}(r)} \quad (1)$$

mit $F := \{(\times, x) \mapsto x, (-, x) \mapsto \neg x, (*, x) \mapsto \mathit{true}\}$. Die Teilformeln $\mathcal{F}_{pre}(r)$ und $\mathcal{F}_{eff}(r)$ heißen Prämissen- bzw. Effektformel.

Eine ET kann wie folgt zur Formalisierung einer Anforderung der Art, daß Systemverhalten in akzeptabel (oder erwünscht) und unerwünscht klassifiziert wird, verwendet werden. Ein System *erfüllt* die durch ET T gegebene Anforderung genau dann, wenn jedes beobachtbare Systemverhalten durch eine Regel in T erlaubt wird. Formal kann eine Beobachtung eines Systems bzgl. der Bedingungen und Aktionen in C und A, also welche Bedingungen aus C erfüllt sind und welche Aktionen aus A ausgeführt werden, als Boole'sche Belegung $\sigma : C \cup A \to \{0, 1\}$ der logischen Variablen in $C \cup A$ notiert werden. Die Beobachtung σ erfüllt T genau dann, wenn es eine Regel r in T gibt, sodaß $\sigma \models \mathcal{F}(r)$.

Begriffe wie Vollständigkeit können für ET präzise definiert werden, z.B. nennen wir eine ET T (formal) vollständig, wenn die Disjunktion der Prämissenformeln von T eine Tautologie ist. Das Problem der Analyse auf (formale) Vollständigkeit kann auf das Erfüllbarkeitsproblem propositionaler Logik reduziert werden, für das in Form von sogenannten SAT-Solvern Entscheidungswerkzeuge zur Verfügung stehen.

In der Vorlesung ‚Softwaretechnik' diskutieren wir nun *nicht* eine Reduktionsprozedur oder die Konstruktion von SAT-Solvern, sondern nehmen eine Nutzersicht ein, nehmen also zur Kenntnis, daß Werkzeuge dieser Art existieren. Wir führen am Beispiel der ET die wichtige Diskussion, wie sich z.B. die formale Vollständigkeit einer ET zur vollständigen Erfassung einer Anforderung im Sinne von (Rupp u. a., 2014) verhält. Eine Analyse einer formalen Beschreibung bzgl. einer Eigenschaft wie Vollständigkeit kann positiv (Eigenschaft nicht gegeben) oder negativ ausfallen und dieses Ergebnis kann (wie Testresultate) falsch oder richtig sein. Ein Grund für ein falschpositives Resultat kann z.B. ein simpler Schreibfehler beim Verfassen der formalen Beschreibung sein. Ein positives Resultat liefert jedoch in jedem Fall ein Indiz dafür, daß uns z.B. in der Anforderungserfassung ein Fehler unterlaufen ist und dieses Indiz ist im Zweifel zusammen mit den Kunden zu klären. Für ET liefern die Werkzeuge im positiv-Fall sogar alle Beobachtungen, die in T nicht berücksichtigt werden, und die sich üblicherweise gut in die Begriffs- und Erfahrungswelt der Kunden übersetzen und als solche klären lassen. Die Formalisierung einer Anforderung und die formale Analyse kann also sicherstellen, daß alle Probleme, die zu positiven Resultaten führen, erkannt und ggf. ausgeräumt werden. Die formale Analyse kann nicht sicherstellen, daß die Anforderungsanalyse als solche vollständig ist, da falsch-negative Resultate möglich sind. Ob der Aufwand der Formalisierung und Analyse von Anforderungen oder Designideen das verringerte Fehlerrisiko rechtfertigt, ist je nach Projektkontext zu entscheiden.

Die Übungsaufgaben zu ET umfassen verschiedene formale Analysen von gegebenen ET sowie eine Aufgabe, in der ein Transkript eines Kundeninterviews (mit absichtlichen Redundanzen und Mehrdeutigkeiten) im Umfang von ca. einer DIN-A4-Seite in eine ET mit ca. 10 Regeln über jeweils 5 Bedingungen und Aktionen überführt werden soll. Im Tutorium stellen wir die (provokante) Frage: Nehmen wir an, man müsste als Implementierer::in des Kundenwunsches zwischen Text und ET als Spezifikation wählen, was wäre die Wahl? Und warum? Meiner Kenntnis nach hat sich in den 4 Jahren der Veranstaltung noch niemand ausdrücklich den Text gewünscht. Hier versuchen wir, einen Aspekt sichtbar zu machen, der unserer Meinung nach im Kontext formaler Beschreibungen noch vor der Anwendung von formalen Analysen steht. Eine formale Beschreibung einer Anforderung hat den Effekt, daß (mit hoher Wahrscheinlichkeit) alle im verwendeten Formalismus ausgebildeten Softwaretechniker::innen zu jeder Zeit dieselbe Information sehen.[8] So sehen wir formale Beschreibungen zuvorderst als Werkzeug, um auf Seiten der Softwaretechniker::innen das Risiko von Missverständnissen zu verringern. Als Analogie verwenden wir z.B. von Jurist::innen konstruierte Individualverträge, deren Konsequenzen man als Nicht-Jurist üblicherweise nicht überblicken kann. Man hat als Auftraggeber des Individualvertrags jedoch den Anspruch, von

[8] Gestandene Praktiker::innen berichten, daß es vorkommt, daß dieselbe Person am selben Tag einen Text verschieden interpretiert.

der Fachperson eine „Übersetzung“ zu erhalten bzw. darauf, wichtige Szenarien gemeinsam durchzuspielen. Ebenso hat man als Softwarekunde einen Anspruch, von den Entwickler::innen deren z.B. in formaler Form notiertes Verständnis der Anforderungen erklärt zu bekommen. Dies kann insbesondere durch Szenarien gelingen (vgl. (Arenis u. a., 2016)). Für jedes von dem/der Kund::in formulierte Szenario können Entwickler::innen auf Basis der formalen Beschreibung Auskunft geben, ob ihrem Verständnis nach dieses Szenario erwünscht oder unerwünscht ist. Bei Verwendung formaler Beschreibungen ist diese Auskunft objektiv und beliebig reproduzierbar.

Die Klausur umfasst Aufgaben verschiedener Taxonomie-Stufen zu ET, vom Beweis z.B. der Vollständigkeit einer ET, über die Analyse einer gegebenen ET auf Konsistenz bis hin zur Erstellung einer kleinen ET zu einem Text. Die Ergebnisse (vgl. (Westphal, 2018)) sind sehr erfreulich. Den gesamten Aufgabenbereich lösten in 2015 bis 2017 mehr als 50 % der Teilnehmer::innen mit 14 bzw. 13 von 15 Punkten, liegen also im guten oder sehr guten Bereich, obwohl wir die Analyseaufgabe klar den „schweren“ Aufgaben der Klausur zuordnen. Die Grenze zum unteren Quartil liegt bei erfreulichen 12,5 bzw. 12 Punkten.

4.2.2 Live Sequence Charts

Als komplexen Formalismus (mit höherer Ausdrucksstärke und Informationsdichte, und höherem Aufwand zur Definition von (abstrakter) Syntax und Semantik) haben wir Live Sequence Charts (Damm u. Harel, 2001) (LSCs) ausgewählt, eine formale Variante der weit verbreiteten Sequenzdiagramme. Wir führen die Automatensemantik für LSCs nach (Klose u. Wittke, 2001) ein, inklusive Chart-Modus, Cold Conditions, und Pre-Charts. Ein System erfüllt eine (universelle) LSC genau dann, wenn alle Berechnungen des Systems von dem aus der LSC konstruierten Automaten akzeptiert werden. Weiterhin führen wir aus, wie LSCs (bzw. ihre Automaten) in der automatischen Testdurchführung verwendet werden können (Klose u. Lettrari, 2001).

Die Klausur umfasst Aufgaben zur Konstruktion der Automaten sowie erfragt Beispiele für akzeptierte bzw. nicht akzeptierte Systemberechnungen. Nicht zuletzt da die Einführung von LSCs den technisch anspruchsvollsten Teil unserer Vorlesung darstellt, sehen wir diese Aufgaben im „mittleren“ und „schweren“ Bereich, der am Ende die guten und sehr guten Gesamtnoten entscheidet. Mit den Resultaten sind wir auch hier zufrieden, das untere Quartil endet bei 8,5 bzw. 7 von 15 bzw. 16 Punkten, das obere Quartil beginnt bei 13 bzw. 12 Punkten.

Im ersten Jahr waren wir von den guten Ergebnissen bei aller Zuversicht positiv überrascht. Wir hatten uns vereinzelt vorgetragenen Sorgen, daß die LSC-Semantik zu schwer sein könnte, nicht völlig entziehen können und entsprechend Vorsorge getragen, die Aufgaben zu LSCs ggf. aus der Wertung zu nehmen.

4.3 Themenbereich Entwurf

Im Themenbereich Softwarearchitektur und Entwurf liegt der Schwerpunkt auf der Modellierung der Struktur und des Verhaltens von Softwaresystemen als Baupläne im Sinne von (Lamport, 2015). Zur Strukturmodellierung führen wir eine übersichtliche Teilsprache der Klassendiagramme von UML ein, die im wesentlichen Klassen mit Attributen von Basistypen und gerichteten Assoziationen der Multiplizitäten 0,1 und $0,*$ mit Ownership umfasst.Neben der bekannten Sicht, Klassendiagramme als Visualisierung von (Klassen in) Programmen zu betrachten, betonen wir die Sicht eines abstrakten (von der Programmierung zunächst unabhängigen) Strukturmodells. Die abstrakte Syntax eines Klassendiagramms ist eine Signatur mit Klassen, Basis- und abgeleiteten Typen, und je Klasse der Menge der Attribute dieser Klasse. Die Semantik eines Klassendiagramms ist die Menge der über dieser Signatur bildbaren Systemzustände (OMG, 2006), d.h. partiellen Funktionen, die (einige) Objektidentitäten auf jeweils eine Belegung der Attribute mit Werten abbilden. Systemzustände können graphisch (ohne Informationsverlust) durch vollständige Objektdiagramme dargestellt werden. Hier stellen wir die Anwendung von Objektdiagrammen in der Dokumentation und Spezifikation heraus. Wenn der Aufwand, bestimmte Annahmen einer Datenstruktur z.B. mit OCL zu formalisieren, in einem Projekt als zu hoch bewertet wird, kann eine geeignete Auswahl von Objektdiagrammen hervorragend zur Spezifikation dieser Annahmen „durch Beispiel“ verwendet werden. Wenn etwa eine Listenstruktur nur unter der Annahme verwendet werden darf, daß keine Zyklen konstruiert werden.

Eigenschaften von Strukturen können mit Hilfe der Object Constraint Language (OCL) formalisiert werden. Hier führen wir Proto-OCL ein, eine Teilsprache der OCL, deren konkrete Syntax sich an der den Studierenden vertrauten Form der Logik erster Stufe orientiert. Die Semantik von Proto-OCL ist eine Interpretationsfunktion, die eine gegebene Proto-OCL-Formel und einen gegebenen Systemzustand (oder ein vollständiges Objektdiagramm) auf einen der drei (!) Werte true, false und $\bot$ abbildet. Wir stellen die Besonderheiten heraus, daß OCL Assoziationen verschiedener Multiplizität verschieden behandelt sowie die Bedingungen, unter denen man eine Auswertung zum dritten Wahrheitswert beobachten kann.

Für die konstruktive Modellierung von Verhalten führen wir eine einfache Teilsprache von State-Machines ein. Unsere *Communicating Finite Automata* (CFA) sind im Wesentlichen nicht-hierarchische State-Machines mit Rendezvous-Synchronisation, d.h. zwei Kanten in verschiedenen Automaten können eine Transition begründen, wenn je eine der Kanten bzgl. eines Events sende- bzw. empfangsbereit ist. Dieses Synchronisationsparadigma ist mit durch State-Machines implementierten

Behavioural Features in UML vergleichbar jedoch nicht (ohne weitere Annahmen) identisch. Zu dieser Abweichung von der UML-Semantik haben wir uns vorrangig aus zwei Gründen entschieden. Einerseits erfordert die formale Einführung des abstrakten Event-Speichers (in der die UML-Semantik bekanntlich parametrisiert ist) sowie einer konkreten Ausprägung, etwa einer empfängerseitigen FIFO-Queue, wie sie IBM Rhapsody verwendet, in unserer UML-Veranstaltung ca. 2 Vorlesungen. Der Erkenntnisgewinn im Vergleich zum Aufwand erscheint uns für eine Softwaretechnik-Veranstaltung zu gering. Andererseits steht mit Uppaal (Larsen u. a., 1997) ein Werkzeug zur Erstellung, Simulation und Verifikation von (insbesondere)[9] CFAs mit einer reichen Programmiersprache für Guards und Aktionen zur Verfügung, das nicht zuletzt aufgrund seines sehr gut auf das Wesentliche reduzierten User-Interfaces[10] hervorragend für unsere Ziele geeignet ist.[11]

Die Übungsblätter zu CFA umfassen Aufgaben zur Erstellung eines Modells eines verteilten Algorithmus für Mutual-Exclusion (das den Teilnehmer::innen als Problem aus der Betriebssysteme-Vorlesung bekannt sein müsste), zur Simulation verschiedener Abläufe und zur Verwendung von Uppaal's Temporallogik (Query Language) zur Spezifikation und Verifikation der Mutual-Exclusion-Eigenschaft. Mit den Übungen verfolgen wir insbesondere das Ziel, das mächtige Modellierungskonzept des Nichtdeterminismus und die durch nebenläufiges Verhalten induzierten Schwierigkeiten sichtbar zu machen.

In der Klausur ist üblicherweise der Transitionsgraph eines gegebenen Systems von mehreren CFAs zu erstellen. Mit den Ergebnissen sind wir durchweg zufrieden, der Median liegt zwischen 10 und 12,25 von 13 Punkten, das obere Quartil beginnt zwischen 12 und 13 Punkten, liegt also im sehr guten Bereich.

4.4 Themenbereich Qualitätssicherung

Im Themenbereich Qualitätssicherung betrachten wir speziell die Prüfung von Programmcode. Hier diskutieren wir zunächst die Disziplin des Softwaretestens. Dem formalen Charakter der gesamten Veranstaltung folgend, führen wir Testfälle als Paare $(In, Soll)$ aus einer (Menge von) Eingabe(n) und Soll-Wert(en) ein. Eine Ausführung eines Testfalls ist eine Beobachtung des Systems, die Eingaben entsprechend In erhält, was ein einzelner Wert oder eine Sequenz von z.B. Events sein kann. Ein Test ist negativ (*Test passed*) genau dann, wenn die betrachtete Beobachtung ein Element von *Soll* ist. Hier betonen wir ((Ludewig u. Lichter, 2013) folgend) besonders den Aspekt, daß die Begriffe des positiven und negativen Tests relativ zu Soll-Werten (die sich ggf. aus (formalen) Anforderungen ableiten lassen) definiert sind. Die Resultate der Klausuraufgaben zum Testen sind durchweg ordentlich, jedoch sind wir jedes Jahr wieder überrascht, daß eine signifikante Anzahl der Studierenden keine Soll-Werte angibt, obwohl explizit nach einer *erfolglosen* Test-Suite gefragt wird.

Als gegenüber den etablierten Lehrbüchern neuen Inhalt präsentieren wir Programmverifikation nach (Hoare, 1969) in der Form des Lehrbuchs (Apt u. a., 2009). Für die Übungen verwenden wir das Web-Interface des Verifying C Compilers (Cohen u. a., 2009) (VCC) und diskutieren als weiterführende Fragen den Unterschied zwischen der Verifikation eines Algorithmus und der Verifikation eines C-Programms unter Annahmen über die Ausführungsplattform. Die Klausuraufgabe besteht üblicherweise darin, eine gegebene Beweisskizze mit den verwendeten Axiomen und Beweisregeln zu annotieren. Hier erreicht das obere Quartil zwischen 5,5 und 7 von 7 Punkten, das untere Quartil endet bei 3 bis 5 Punkten.

5 Lehrevaluation

Da die Veranstaltung neu konstruiert wurde, beobachten wir seit der ersten Durchführung verschiedene Metriken (vgl. Abschnitt 4.1) auf Indizien für Fehlentwicklungen. Daten beziehen wir aus dem Übungsbetrieb, den Klausuren und der zentralen Evaluation.

In der Evaluation beachten wir besonders die Aspekte „Workload“ und „Niveau“ (Uttl u. a., 2017) und die Freitexte. Den Workload sehen über alle 4 Jahre gesehen rund 62 % der Studierenden als ‚mittel‘ und rund 30 % als ‚hoch‘, das Niveau sehen rund 62 % als ‚angemessen‘ und rund 29 % als ‚hoch‘. Dies entspricht vollständig unserem Ziel für eine universitäre Lehrveranstaltung.

Den Lernerfolg bzgl. des Lernziels der Vermittlung von Methoden und Techniken messen wir anhand der Klausurergebnisse, mit denen wir über alle 4 Jahre sehr zufrieden sind. Das viel interessantere Lernziel der Vorbereitung auf das „echte Leben“ läßt sich leider notorisch nicht gut messen. Als ein positives Indiz sehen wir unsere und von Kolleg::innen berichtete Erfahrung aus Vorgesprächen zu individuellen Projekten wie etwa B. Sc.-Arbeiten. Hier nehmen wir die Inhalte der Vorlesung als (angestrebten) soliden Bezugspunkt wahr, um über spezialisierte Aufgabenstellungen zu sprechen. Wir hoffen, daß sich die Veranstaltung ebensogut als Bezugspunkt für den Einstieg in die berufliche Karriere der Studierenden eignet.

6 Zusammenfassung

Formale Methoden werden zunehmend in vielen Bereichen der Softwaretechnik angewandt. Unserer Überzeugung nach ist es hilfreich, wenn die Bedeu-

[9] Daß Uppaal für CFAs mit Uhrenvariablen, also Timed Automata, entwickelt wurde, verschweigen wir schlankerhand.

[10] Aus denselben Überlegungen führen wir kein Werkzeug zur Erstellung von Klassendiagrammen ein. Die für die Einarbeitung in die Bedienung der meisten dieser Werkzeuge notwendige Zeit steht unserer Meinung nach in keinem akzeptablen Verhältnis zu den Absichten z.B. unserer Übungen (vgl. (Glinz, 1996)).

[11] Einige Teilnehmer::innen unserer UML-Veranstaltung, in der wir IBM Rhapsody verwenden, vermissen die mächtige und leicht zu bedienende Simulationsfunktion von Uppaal.

tung formaler Beschreibungen vollständig durchdrungen wird und die aus formalen Analysen zu ziehenden Schlüsse verstanden werden.

Unsere Erfahrung zeigt, daß es möglich ist, eine Veranstaltung zur Einführung in die Softwaretechnik im Grundstudium um genau diese Aspekte zu ergänzen ohne hierbei die Inhalte der etablierten Lehrbücher zu vernachlässigen.

Danksagungen. Wir danken Sergio Feo Arenis und Christian Schilling für ihre Unterstützung als Assistenten der ersten beiden bzw. des letzten Jahres. Ohne die Gespräche mit Sergio und Christian und ihre Ausarbeitung und Weiterentwicklung der Übungsaufgaben wäre die Veranstaltung in ihrer heutigen Form nur schwer vorstellbar. Weiterhin danken wir unseren stud. Hilfskräften für ihre engagierte Mitarbeit an Aufgaben und Tutorials.

Literatur

[Anderson u. a. 2001] ANDERSON, L. W. (Hrsg.) u. a.: *A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, 2001

[Apt u. a. 2009] APT, K. R. ; BOER, F. S. ; OLDEROG, E.-R.: *Verification of Sequential and Concurrent Programs*. Springer, 2009

[Arenis u. a. 2016] ARENIS, S. F. ; WESTPHAL, B. u. a.: Ready for testing: ensuring conformance to industrial standards through formal verification. In: *FAoC* 28 (2016), Nr. 3, S. 499–527

[Balzert 2009] BALZERT, H.: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. 3rd. Spektrum, 2009

[Biggs u. Tang 2011] BIGGS, J. ; TANG, C.: *Teaching for Quality Learning at University*. 4th. Open University Press, 2011

[Bjørner 2006a] BJØRNER, .: *SWE, Vol. 1: Abstraction and Modelling*. Springer, 2006

[Bjørner 2006b] BJØRNER, D.: *SWE, Vol. 2: Specification of Systems and Languages*. Springer, 2006

[Bjørner 2006c] BJØRNER, D.: *SWE, Vol. 3: Domains, Req. and Software Design*. Springer, 2006

[Bloom 1956] BLOOM, B. S. (Hrsg.): *Taxonomy of Educat. Objectives: Cognitive Domain*. Longman, 1956

[Cohen u. a. 2009] COHEN, E. u. a.: VCC: A Practical System for Verifying Concurrent C. In: BERGHOFER, S. (Hrsg.) u. a.: *TPHOLs* Bd. 5674, Springer, 2009 (LNCS), S. 23–42

[Damm u. Harel 2001] DAMM, W. ; HAREL, D.: LSCs: Breathing Life into Message Sequence Charts. In: *FMSD* 19 (2001), Juli, Nr. 1, S. 45–80

[Glinz 1996] GLINZ, M.: The Teacher: "Concepts!" The Student: "Tools!". In: *Softwaret.-Trends* 16 (1996), Nr. 1

[Hoare 1969] HOARE, C. A. R.: An axiomatic basis for computer programming. In: *CACM* 12 (1969), Nr. 10, S. 576–580

[IEEE 1990] IEEE: *Standard Glossary of Software Engineering Terminology*, 1990. – Std 610.12-1990

[IEEE 1998] IEEE: *Recommended Practice for Software Req. Specifications*, 1998. – Std 830-1998

[ISO/IEC/IEEE 2010] ISO/IEC/IEEE: *Systems and software eng. – Vocabulary*, 2010. – 24765:2010(E)

[Klose u. Lettrari 2001] KLOSE, J. ; LETTRARI, M.: Scenario-based Monitoring and Testing of Real-time UML models. In: GOGOLLA, M. (Hrsg.) u. a.: *UML* Bd. 2185, Springer, 2001 (LNCS)

[Klose u. Wittke 2001] KLOSE, J. ; WITTKE, H.: An Automata Based Representation of Live Sequence Charts. In: MARGARIA, T. (Hrsg.) u. a.: *TACAS*, Springer, 2001 (LNCS 2031)

[Krusche u. a. 2017] KRUSCHE, S. ; FRANKENBERG, N. von ; AFIFI, S.: Experiences of a Software Engineering Course based on Interactive Learning. In: *SEUH*, 2017, S. 32–40

[Lamport 2015] LAMPORT, L.: Who builds a house without drawing blueprints? In: *CACM* 58 (2015), Nr. 4, S. 38–41

[Langenfeld u. a. 2016] LANGENFELD, V. ; POST, A. u. a.: Requirements Defects over a Project Lifetime. In: DANEVA, M. (Hrsg.) u. a.: *REFSQ* Bd. 9619, Springer, 2016 (LNCS), S. 145–160

[Larsen u. a. 1997] LARSEN, K. G. ; PETTERSSON, P. ; YI, W.: UPPAAL in a Nutshell. In: *STTT* 1 (1997), Dezember, Nr. 1, S. 134–152

[Lehmann u. a. 2015] LEHMANN, T. u. a.: Lecture Engineering. In: SCHMOLITZKY, A. (Hrsg.) u. a.: *SEUH* Bd. 1332, CEUR-WS, 2015, S. 103–109

[Ludewig u. Lichter 2013] LUDEWIG, J. ; LICHTER, H.: *Software Engineering*. 3rd. dpunkt, 2013

[OMG 2006] OMG: *Object Constraint Language, Version 2.0*. formal/06-05-01, 2006

[Rupp u. a. 2014] RUPP, C. u. a.: *Requirements-Engineering und -Management*. 6th. Hanser, 2014

[Sedelmaier u. Landes 2017] SEDELMAIER, Y. ; LANDES, D.: Experiences in Teaching and Learning Req. Engineering on a Sound Didactical Basis. In: DAVOLI, R. (Hrsg.) u. a.: *ITiCSE*, ACM, 2017, S. 116–121

[Siegeris 2017] SIEGERIS, J.: LearnTeamPlenum. In: *SEUH*, 2017, S. 1–7

[Sommerville 2010] SOMMERVILLE, I.: *Software Engineering*. 9th. Pearson, 2010

[Uttl u. a. 2017] UTTL, B. u. a.: Meta-analysis of faculty's teaching effectiveness: Student evaluation of teaching ratings and student learning are not related. In: *Stud. Educat. Eval.* 54 (2017), S. 22 – 42

[Westphal 2018] WESTPHAL, B.: An Undergraduate Requirements Engineering Curriculum with Formal Methods. In: *REET*, IEEE, 2018, S. 1–11

Stager: Simplifying the Manual Assessment of Programming Exercises

Christopher Laß, Stephan Krusche, Nadine von Frankenberg, Bernd Brügge
Technische Universität München
christopher.lass@tum.de, krusche@in.tum.de, nadine.frankenberg@in.tum.de, bruegge@in.tum.de

Abstract

Assessing programming exercises requires time and effort from instructors, especially in large courses with many students. Automated assessment systems reduce the effort, but impose a certain solution through test cases. This can limit the creativity of students and lead to a reduced learning experience. To verify code quality or evaluate creative programming tasks, the manual review of code submissions is necessary. However, the process of downloading the students' code, identifying their contributions, and assessing their solution can require many repetitive manual steps.

In this paper, we present Stager, a tool designed to support code reviewers by reducing the time to prepare and conduct manual assessments. Stager downloads multiple submissions and adds the student's name to the corresponding folder and project, so that reviewers can better distinguish between different submissions. It filters out late submissions and applies coding style standards to prevent white space related issues. Stager combines all changes of one student into a single commit, so that reviewers can identify the student's solution more quickly.

Stager is an open source, programming language agnostic tool with an automated build pipeline for cross-platform executables. It can be used for a variety of computer science courses. We used Stager in a software engineering undergraduate course with 1600 students and 45 teaching assistants in three separate programming exercises. We found that Stager improves the code correction experience and reduces the overall assessment effort.

1 Introduction

The number of students in university courses is increasing. The number of new undergraduate students at our computer science department increased by 81 % between 2013 (1110 students) and 2017 (2005 students)[1]. Practical programming exercises are essential in computer science education and help students acquire important skills in software development [Staubitz et al., 2015]. However, a manual assessment of programming exercises in large courses can take a considerable amount of time and effort. Automatic assessment systems (also called auto-graders) aim at flexibility and scalability in large courses, and allow to integrate exercises into lectures [Krusche et al., 2017b]. These systems utilize, among others, version control systems (VCS) to store the code solutions of students in repositories and test cases that are executed on a continuous integration server to assess the solution to a programming exercise automatically [Heckman and King, 2018; Krusche and Seitz, 2018].

While automated assessment systems significantly reduce manual assessment effort, they have drawbacks. Predefined test cases cannot cover all possible solutions and therefore impose a certain solution on the students. Some students are limited in their programming skills, while other students can exploit the test cases by repetitive trial-and-error submissions. Especially first year students who are new to programming often experience problems when trying to formulate their solution and thoughts as an executable computer program [Robins et al., 2003]. Such submissions can be overly complicated, and assessment systems cannot (yet) provide enough useful feedback in that regard. Furthermore, some programming exercises cannot be assessed automatically. The automated grading of creative assignments with open problem statements is hardly possible because different solutions exist [Knobelsdorf and Romeike, 2008; Krusche et al., 2017a]. An example for such an assignment is to implement a creative collision strategy in a 2D racing game. Automated test cases could be able to validate a collision, but are incapable of assessing the creativity or code quality of the solution. As a result, manual assessment can be beneficial, even in large courses that have fully implemented automated grading solutions.

However, the process of manually assessing multiple students' solutions requires repeated manual steps. Tasks such as finding the next student's repository, downloading the source code, and renaming the folders and projects names for standardization can be time-consuming and error-prone. Determining a student's contribution is challenging when the exercise builds upon a provided code template and when the

[1] https://www.tum.de/die-tum/die-universitaet/die-tum-in-zahlen/studium

students use multiple commits in their code repository. Then it becomes difficult to separate the provided template and the final solution.

In this paper, we present Stager, a tool that is designed to support the manual assessment of programming exercises. Reviewers, e.g. teaching assistants or instructors, can automate the manual steps that are necessary to prepare the students' code repositories, for instance download all repositories at once, and thereby reduce the manual assessment time. The idea for Stager evolved during an undergraduate university course with 1600 students and 45 teaching assistants. An initial implementation was used for three separate programming assignments.

The remainder of the paper is organized as follows. We describe related work focusing on existing automated assessment solutions and the limitations of automated assessment approaches in Section 2. In Section 3, we cover Stagers' approach to automating the recurring manual steps during the correction of programming exercises. We describe design decisions, the exercise workflow with Stager, the configuration possibilities of the tool, and the concrete tasks of Stager, e.g. the *Download repositories* task. We analyze the improved code assessment experience of the teaching assistants by means of an experience report in Section 4, where we also present the results of a quantitative analysis of Stager's use in three programming exercises. Section 5 concludes the paper and provides directions for future work.

2 Related Work

Several automated assessment system approaches for programming assignments exist [Heckman and King, 2018; Knobelsdorf and Romeike, 2008; Krusche and Seitz, 2018; Pieterse, 2013]. Advantages include a decrease in the workload of course instructors and timely feedback for students [Pieterse, 2013]. Automated systems work well to grade programming assignments consistently and evaluate specific aspects, e.g. the functionality [McCracken et al., 2001] or efficiency of a system [Jackson and Usher, 1997]. However, they are missing the benefit of personal feedback which a manual grading approach could provide. The test cases used by such systems cannot assess the code quality and "elegance" of the solution [Poženel et al., 2015].

Building a robust automated assessment system amounts to a heavy workload, whereby the definition of the test cases is (usually) the most time consuming activity [Cerioli and Cinelli, 2008]. This workload is amplified when designing tasks with some degree of freedom of solutions [Chen, 2004]. The degree of freedom of solutions indicates the difficulty of the exercise [Striewe and Goedicke, 2013], meaning that a difficult exercise has more possible solutions and therefore has an increased workload to design the automated assessment system. Depending on the class size, it can therefore be less time consuming to manually assess solutions rather than to design the automated assessment system [Ala-Mutka, 2005].

Further, students can become distracted by automated feedback. For instance, students may be tempted to fix only the failing tests instead of focusing on the assignment [Heckman and King, 2018]. Automated assessment systems circumvent the detection of frequent mistakes or misunderstandings among students. The understanding and resolution of common errors is an essential learning experience for students. Semi-automated systems combine the mentioned aspects by providing automated grading, as well as manual feedback. Such systems offer personalized feedback to some extent, for instance the instructor can annotate a static assessment [Gerdes et al., 2017]. Other systems give the student instant feedback if the student's solution is correct. If it is not, the instructor reviews each solution and can give additional feedback if required [Insa and Silva, 2015]. Many systems focus on the grading itself, but not on the process the instructor has to follow to obtain the students' solutions.

Some commercially available systems and tools that are used in computer science (CS) courses offer features that aim at simplifying this process. In 2000, Jackson proposed an approach that pre-processes student submissions (sent via e-mail) by removing irrelevant information or unpacking files [Jackson, 2000]. For submissions via repositories, pull requests (also called merge requests) in GitHub[2], GitLab[3], or Bitbucket[4] allow students to commit their changes into separate branches. After requesting the code to be merged into the main branch, i.e. a submission, reviewers can highlight the student's contribution as difference to the template code and provide feedback by requesting changes. While pull requests can also be integrated with continuous integration systems, e.g. using TravisCI[5] to detect compile errors and to run automated tests, reviewers might still need to download the source code and execute it to verify if all requirements of the problem statement have been solved.

GitLab introduced a "Squash and Merge" option which "applies all of the changes in a merge request as a single commit, and then merges that commit using the merge method set for the project"[6]. This cleans up the commit history and can make it easier to identify the contribution of one particular student. Tools and services, such as Gerrit[7], support code reviews that enable the reviewer to see the code difference, and provide the option to leave in-line comments.

[2] https://github.com
[3] https://gitlab.com
[4] https://bitbucket.org
[5] https://travis-ci.org
[6] https://docs.gitlab.com/ee/user/project/merge_requests/squash_and_merge.html
[7] https://www.gerritcodereview.com

However, such tools primarily focus on continuous feedback rather than assessing a student's solution.

3 Stager's Approach

This section presents an approach that automates manual steps during the correction of programming exercises in order to prepare student repositories for easier assessment. We show how code reviewers can use Stager. Furthermore, we explain the different tasks that are automatically executed by Stager.

Figure 1 illustrates the exercise workflow including the manual assessment with the help of Stager as a UML activity diagram. As precondition for this workflow, every student must have their own repository with the code template for the exercise in a VCS[8]. After the students complete the exercise, they commit and push their solutions to the VCS (action 1.3).

Before reviewers start to work, they need to configure Stager (action 2.1). Then, they trigger Stager to process different tasks (actions 3.1 ... 3.6), such as *Download repositories* or *Normalize code style*. Finally, the reviewer can manually assess the pre-processed submissions and give qualitative feedback (action 4.2 and 5.) to the students in any arbitrary form (e.g. uploading the feedback into an exercise management system such as Moodle[9]).

The action *2.1 Configure Stager* of the *Reviewer* is described in Section 3.1. *Stager*'s actions are described as tasks in Section 3.2. The numbering in Section 3.2 aligns with the corresponding action in Figure 1.

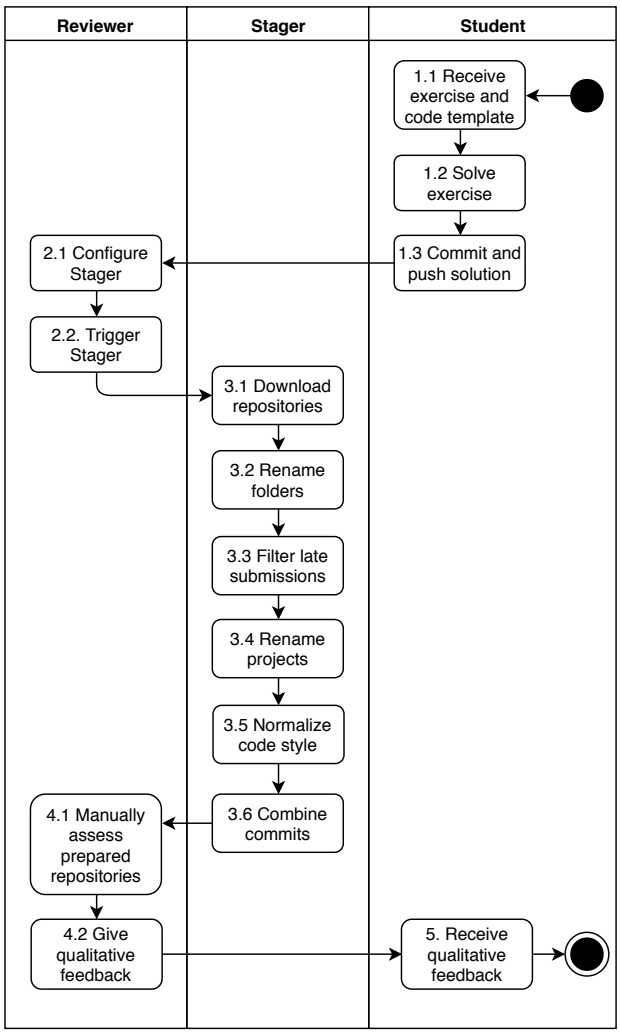


Figure 1: Exercise workflow with Stager: students complete the exercise and upload their solutions to a VCS. The reviewer configures and triggers Stager to process different tasks, e.g. *3.1 Download repositories*. Afterwards, the reviewer manually assesses the prepared repositories and gives qualitative feedback to the students.

3.1 Stager's Setup

Stager is free, open source, and available under the MIT license[10]. It is platform independent and programming language agnostic, making Stager universally applicable. It is written in the Go programming language[11] and makes use of the distributed version control system git[12]. Cross-platform executables can be downloaded from the automatic build pipeline or compiled from the source code.

Stager's configuration is separated into two files *students.csv* and *config.json*, based on how frequently the settings change. The list of students in *students.csv* might not change during the course duration, while *config.json* changes for every exercise. The configuration procedure must be completed after the code template was finished and before Stager is executed. Stager or its configuration does not add any preconditions or constraints on the students. The following settings can be edited:

1. Credentials: Remote git repositories can be accessed via the SSH or HTTP protocols [Lawrance et al., 2013]. For HTTP, the JSON keys *username* and *password* have to be set with valid credentials and access rights to the VCS. For SSH, Stager uses the operating system's global SSH settings and therefore does not require further configuration.

2. Latest commit hash of a programming exercise template: The programming exercises that are distributed to the students build upon a given code template. The SHA hash of the latest commit for the code template, meaning the latest code changes the reviewer included, must be set for the JSON key *squash_after*. This setting is required for Stager to distinguish between the given code by the reviewer and code written by the student. This configuration option is used by the task *Combine commits* and is further elaborated in Section 3.2.

3. Deadline for homework submission: Students have to submit their homework in a given time-frame.

[8] There are multiple tools available that automate this step, e.g. ArTEMiS, Github Classroom, etc.

[9] `https://moodle.org`

[10] `https://github.com/arubacao/stager`

[11] `https://golang.org`

[12] `https://git-scm.com`

For example, the homework must be submitted by Sunday midnight because the programming exercises will be discussed in class on Monday morning. However, VCSs have limitations when it comes to time-based repository access. As described in more detail in Section 3.2, the task *Filter late submissions* allows to overcome these VCSs limitations. The deadline for students submitting their homework is set with the JSON key *deadline*. The standard datetime format `YYYY-MM-DD HH:MM:SS` must be used. For example, `2018-08-31 23:59:59` is valid.

4. Remote repository URL schema: Each student has a personal repository that can be accessed with a unique URL. A general URL schema can be derived from these unique URLs, where the students' identifiers are substituted by a placeholder. For example, for the repository URL (1) of student *10001*, the derived general URL schema is (2). If the repositories are accessed using HTTP as in the example, two additional placeholders must be set for the reviewer's credentials (3). The resulting schema is set for the key *url*.

*https://repo.uni/cs101/exercise01-**10001**.git* (1)

*https://repo.uni/cs101/exercise01-**%s**.git* (2)

*https://**%s:%s**@repo.uni/cs101/exercise01-**%s**.git* (3)

5. List of students: In addition to the mentioned settings, Stager requires a list of students the reviewer wants to assess. The students' names and identifiers are defined in the *students.csv* file with the format shown in Listing 1. All mentioned people and courses in this paper are placeholder names and do not exist in reality.

Listing 1: Sample students.csv

```
name,id
John Doe,10001
Jane Roe,10002
```

After configuration, the Stager executable, *config.json*, and *students.csv* are placed in a dedicated and empty folder. Stager can then be executed via a double click or from the terminal. Listing 2 illustrates this workflow. After Stager terminates, the students' repositories are locally available and prepared by the tasks described in the following Section 3.2.

Listing 2: Folder setup and execution of Stager

```
$ cd ~/cs101/assessment3
$ ls
config.json   stager   students.csv
$ ./stager
```

3.2 Stager's Tasks

Stager provides an extendable framework which makes it easy to add or remove tasks according to the reviewer's requirements. Tasks are functions that modify the repository or its contents and have a single purpose. For example, the *Rename folders* task appends the student's name to the corresponding folder. Stager is composed of multiple tasks (shown in the Stager swimlane in Figure 1) that adhere to certain rules and are sequentially performed during the tool's execution. The implementation allows a clear distinction of tasks, such that each task addresses a separate purpose. Therefore, it is easy to add new tasks or remove existing ones conceptually and implementation-wise in the future. For example, when the reviewer does not need a certain task, only one line of code within the array of tasks has to be removed. Furthermore, tasks must be idempotent, meaning that multiple executions of the task lead to the same output. Even though tasks are independent, they are processed sequentially, i.e. the order of the tasks is relevant. For instance, repositories first have to be downloaded before other tasks have local file access.

The goal of Stager is to simplify the manual assessment of programming exercises by modifying source code, files, and repositories. Repetitive manual steps that are required for the reviewer to start the assessment should be reduced or eliminated by Stager. We identified the following relevant tasks (listed according to the order of execution) and describe each of them in detail in the following:

1. Download repositories
2. Filter late submissions
3. Rename folders
4. Rename projects
5. Normalize code style
6. Combine commits

1. Download repositories: In order to better determine the software quality and verify if all requirements of the problem statement have been solved by the students' submissions, it is necessary for the reviewer to compile and execute their homework source code locally. Hence the repositories must be available on the reviewer's computer. The initial task clones all repositories of the predefined students *as-is* and *all at once* to a given folder on the reviewer's computer. This first task takes potential existing local repositories into account and overwrites them. It ensures that each local repository is in sync with the remote repository and in a clean state.

The following tasks modify files and therefore require write access to the repositories. These modifications can only be performed when the repositories are locally available. Consequently, the *Download repositories* task must be first.

2. Filter late submissions: Homework submissions are tied to a hard deadline. With web-based VCSs like Bitbucket or GitLab, it is hardly possible to block student commits after a given deadline. Students could exploit this situation and extend their

time to finish the exercise as shown in Figure 2. The *Filter late submissions* task analyzes the commit timestamps and sets the repository to the state of the pre-configured deadline in *config.json*. Commits after the deadline are not considered anymore. This way time-based limitations of web-based VCSs are bypassed. However, this procedure is not fully forgery-proof, since commit timestamps can be manipulated.

File changes made prior to this task would be striped out, since the repository is set to the state of the pre-configured deadline. Therefore, the *Filter late submissions* task must be executed before any other task can modify files.

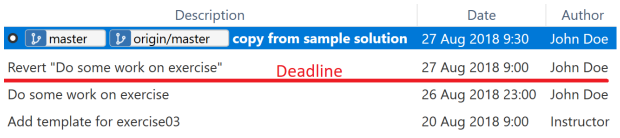


Figure 2: Filter late homework submissions by excluding commits after the homework submission deadline. The two commits above the red line are after the deadline while the two commits below the red line are before the deadline.

3. Rename folders: Depending on the naming convention, only the student's identifier is used for the repository name. The resulting folders can be hard to keep separate and to associate with the correct student. For obfuscation and identity protection this is reasonable, but counterproductive on the reviewer's local system since it is easier to identify a student by their name and not through their id. Once the repositories are locally available, the *Rename folders* task appends the student's name to the corresponding folder as illustrated in Figure 3.

```
1    - cs101-exercise03-10001
2    - cs101-exercise03-10002
1    + cs101-exercise03-10001_john_doe
2    + cs101-exercise03-10002_jane_roe
```

Figure 3: Append names to folders to better distinguish between students. Without the names john_doe and jane_roe, it would be difficult to identify which folder belongs to which student.

4. Rename projects: As precondition of Stager, each student must have their own repository for each published exercise. The content of these repositories is always identical. As a result, the project names are also identical for all students. This leads to the problem that reviewers could only import one project at the same time into Eclipse in order to review and execute the code. Renaming all projects manually is time-consuming and error-prone. Analogue to the *Rename folders* task, a student's name is prepended to the corresponding project name. This makes it possible to distinguish between students within source code editors or integrated development environments (IDEs), e.g. Eclipse[13] (Figure 4), and allows to import multiple projects at the same time. Eclipse, for instance, does not allow to import multiple projects with identical names, which makes it impossible to compare multiple solutions without renaming the projects.

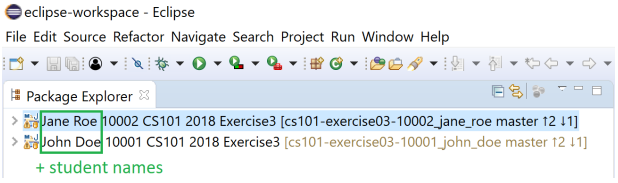


Figure 4: Prepend student names to projects so that the submissions of multiple students can be imported into Eclipse and reviewed at the same time. Jane Roe and John Doe are prepended to the project name. Otherwise the reviewer could only import one Eclipse project at the same time.

5. Normalize code style: The encoding and code style of the provided code template and the final student's contribution should be consistent. Windows and Unix-based systems use different line breaks for code files by default. Windows uses carriage return and line feed ''\r\n'' as a line ending, whereas Unix based systems use just line feed ''\n''. Also, IDEs might automatically enforce a different code style standard than desired. As illustrated in Figure 5, this could lead to non-relevant changes and obscured code differences in commits, thereby making it harder to assess the submission. To avoid these non-relevant file changes by the student, Stager invokes a *linter* that automatically normalizes the code to the same standards as the initial template. This means that all white space related changes, e.g. line breaks, empty spaces and tabs are removed, so that the reviewer does not need to analyze them. Each programming language has its own linting strategies, utilizing existing tools like *eslint*[14] for Javascript or *checkstyle*[15] for Java. This hides pure white space and encoding changes and allows code reviewers to focus on the actual contributions by the students.

6. Combine commits: Reviewers provide code templates as a starting point for the programming exercise, in which the student has to make changes across multiple files. These changes can be small compared to the provided template and consequently hard to identify by the reviewer. In order to determine the student's contribution more effectively, it is helpful to see the exact difference between the template and the final submission instead of only looking at the final submission. VCSs provide easy comparison methods

[13] https://www.eclipse.org
[14] https://github.com/eslint/eslint
[15] https://github.com/checkstyle/checkstyle

```
1   - public int getSpeed(){
2   -       return this.speed;
3   - }
1   + public int getSpeed(){
2   +       return this.speed;
3   + }
```

Figure 5: There is no visual change in the two code blocks in this figure. However, non-visible *line breaks* cause the comparison tool to show these lines. This can make it time-consuming for the reviewer to identify relevant changes.

where the difference made by a single commit is visible. However, a submission can consist of multiple commits. The reviewer would have to compare each commit and memorize the changes themselves, which makes the standard comparison method impractical and error-prone.

The *combine commits* task combines the students commits into one single commit. The reviewer does not need to review multiple changes within the same code line and can omit changes that have been added in one commit and removed again in a later commit. This single commit also contains all Stager related changes (e.g. white space changes). As a result, it is easy for the reviewer to quickly identify the student's contribution and to decide if the solution is correct. In addition to the existing branches with the complete commit history, Stager adds the combined commit into a separate branch. Thus, information is only added and not removed from the repository and the reviewer could still see the whole commit history. Web-based VCSs like GitHub also offer a squash feature, however, the reviewer would have to trigger it manually for each repository.

Figure 6 illustrates this process with an example student *John Doe* and an *Instructor*. The *Instructor* provides a code template. *John Doe* works on the given exercise. Over a period of one day, *John* submits his work separated across multiple commits. As seen in the bottom right corner of Figure 6, one assignment was to *Add new car types to the game*. Since *John* submitted multiple code changes and removed the "TODO" lines within the code, the reviewer would have to actively scan all nine commits to identify *John*'s solution. Stager solves this time-consuming process by combining all student commits into one single commit that includes all changes by *John*. This single commit is selected in the top of Figure 6. The reviewer can see every file that has been modified by the student and quickly identify, whether *John* has completed the assignment correctly.

4 Experience Report

The following experience report describes the lecture-based course Introduction to Software Engineering (EIST[16]) in which we used Stager to improve the manual assessment of programming exercises. EIST is a second semester bachelor's course with a heterogeneous group of students including computer science, business informatics, and business students.

The course assumes that students have successfully completed an introductory course in computer science (e.g. CS1) and are familiar with object-oriented programming in Java. The course's learning goals are that students are able to apply relevant concepts and methods in all phases of software engineering projects including analysis, design, implementation, testing, and delivery. Further, students know the most important terms and concepts and can apply them in modeling and programming tasks. They are aware of the problems and issues that generally have to be considered in software engineering projects. Table 1 shows the schedule and the content of the course.

Week	Content
1	Introduction
2	Model-Based Software Engineering
3	Requirements Elicitation and Analysis
4	System Design I
5	System Design II
6	Object Design
7	Model Transformations and Refactorings
8	Pattern-Based Development
9	Lifecycle Modeling
10	Software Configuration Management
11	Testing
12	Project Management
13	Repetitorium

Table 1: The course *Introduction to Software Engineering* lasts 13 weeks.

1600 students were registered for the course in 2018. One lecturer and three exercise instructors were involved in the organization of the course. 45 teaching assistants were responsible for holding 74 exercise group sessions per week. Teaching assistants were mainly bachelor students in the fourth semester, who successfully completed the same course in the previous year.

The course design is based on interaction and assumes active participation from students. The interactive parts include in-class exercises, in-class quizzes, and exercise sessions. Students need to bring their laptops to the class and to exercise sessions. Students can earn bonus points for completing in-class and homework exercises successfully. They can use these bonus points to improve their final exam grade.

[16]The German title is "Einführung in die Softwaretechnik".

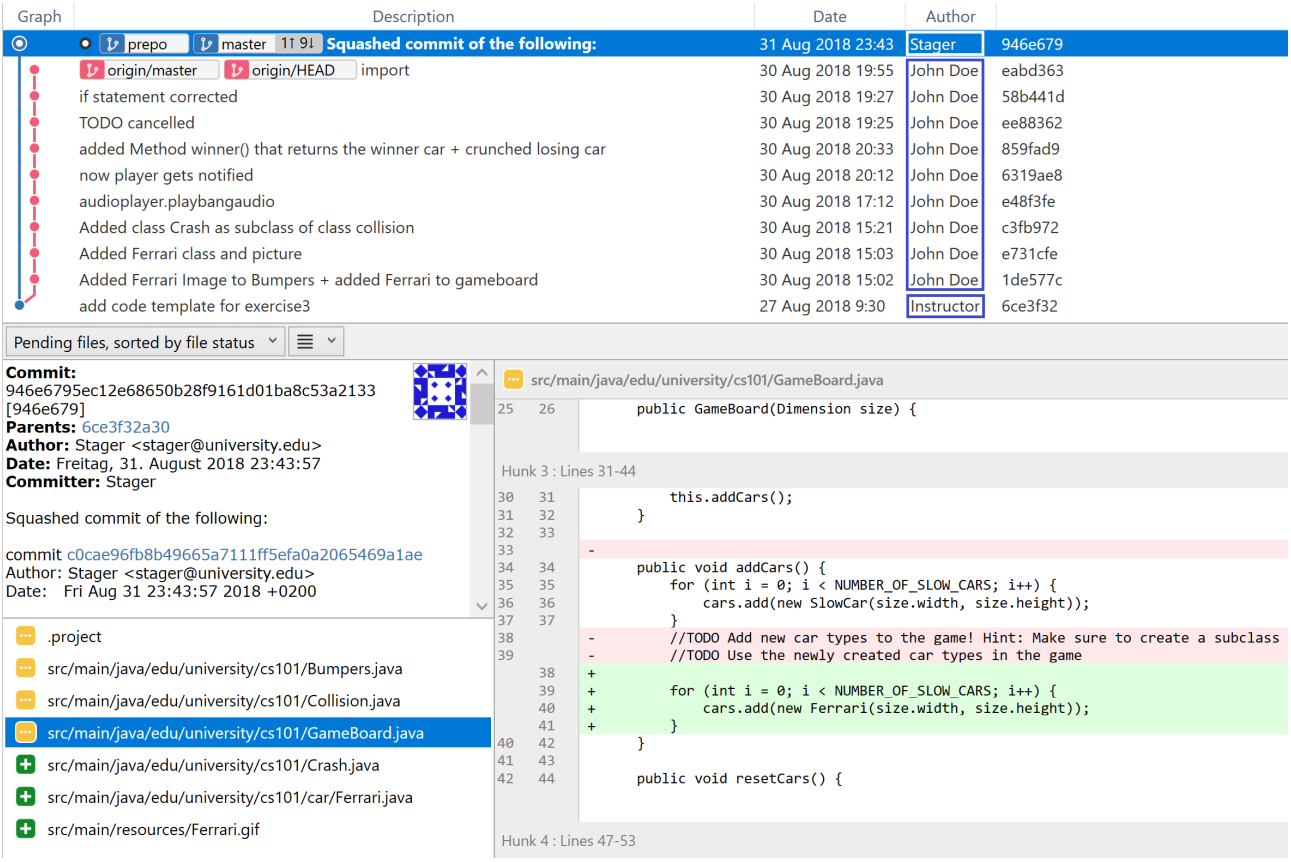


Figure 6: Student commits are combined into one discrete change set: the commit at the top highlighted in blue. This commit displays the difference between a provided code template by the instructor and the submitted solution by the student. All commits of the student John Doe are still available.

For instance, if they score more than 90 % of the total exercise points, their grade in the final exam is improved by 1.0. This possibility motivates the students to participate in the in-class exercises and in the homework exercises. In-class exercises consist of quizzes (similar to the quiz exercises described in [Krusche et al., 2017c]), modeling and programming exercises. Homework exercises include modeling, text and programming exercises.

4.1 Programming Exercises

Between 600 and 1200 students have actively participated in each programming exercise throughout the semester which is shown in Figure 7 and Figure 8. In each exercise, the students had to write new source code or adjust existing code based on a given problem statement. All students worked on the existing template code of an exercise in their individual git repository. The exercises were based on a 2D racing game called Bumpers. In the game, cars collide with each other and each collision has a winner. The course is designed so that each week's exercises focus on a different part of Bumpers in accordance with the lecture's content, e.g. in week 8, "Pattern-Based Development", exercises include the implementation of different design patterns to make the game extensible for new requirements.

To submit their solutions, the students commit their changes to a version control system. This automatically triggers test cases on a continuous integration server to verify the given solution. After the submission of their solution, students can automatically see the test results as individual feedback and improve their solution according to this feedback.

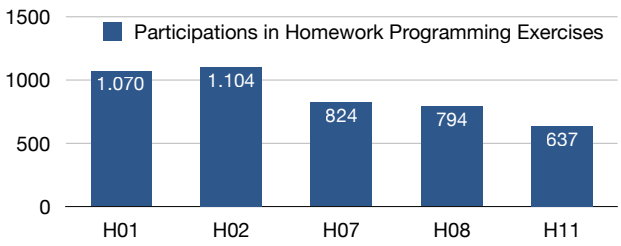


Figure 7: Number of students who submitted solutions to homework programming exercises

However, not all aspects of a problem statement can be automatically tested. Either it is difficult to test a certain aspect of a solution, for instance complex behavior tests, or the problem statement provides a

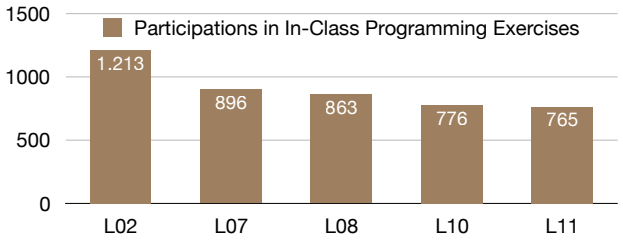


Figure 8: Number of students who submitted solutions to in-class programming exercises

high degree of freedom which makes it difficult to write test cases, e.g. open or visionary questions.

The following three homework programming exercises required manual assessment by the teaching assistants. The second and third exercises were graded semi-automated.

1. Collision Detection: The task was to implement a creative collision detection algorithm for cars in Bumpers. The students were given executable template code and had to extend it with a new class that included their solution. This exercise required manual correction to test whether the new collision algorithm performed as intended. Additionally, the most creative solutions were awarded and shown in class.

2. Serialization of Code: The students had to instantiate objects from two classes in Java. The main task was to serialize and deserialize these object using JSON. An automated assessment system was used to test the input and output of the serialization. However, the students wrote their own serialization code, so their solutions varied, e.g. in the naming of the objects or methods. This required the teaching assistants to assess the implementations manually.

3. Adapter Pattern: Based on a code template, the assignment was to extend the 2D car racing game Bumpers with legacy code using the adapter pattern. The legacy code for an existing analog speedometer panel was provided separately. An automated assessment system graded the students' solution. In addition, the teaching assistants had to verify if the speedometer panel was shown in the game user interface and displayed the velocity correctly.

4.2 Results

In order to determine how many manual steps during a homework assessment can be automated by Stager, we conducted a quantitative analysis for these three programming exercises. For the qualitative analysis we focused on:

1. Number of commits per student
2. Number of commits after the exercise deadline
3. Source code changes where only white spaces have been added or removed

Table 2 displays an overview of the number of participating students for each exercise together with submission metrics. The number of commits per student varies from 1.81 to 5.91 on average. Stager's *combine commits* task will combine student commits into one single commit so that reviewers can distinguish the difference between the provided code template and code submitted by the student immediately. There are respectively 34, 8, and 7 late submissions for the observed exercises. Stager will automatically filter commits that are contributed after the defined exercise deadline. There are between 118 and 183 students that submitted at least one commit where they only changed white spaces. While reviewing the student contributions, white space related changes are visually distracting to the reviewer (see Figure 5), since these changes are not relevant to the exercise.

In informal discussions, seven teaching assistants reported that Stager reduced their reviewing effort significantly. The workflow without Stager required the teaching assistants to first filter the repositories by student, then to check the commit dates and times, clone or download the code, and to fix potential white space problems in order to be able to assess the actual sumbission. Depending on the amount of exercise sessions, teaching assistants had to perform this manual workflow for up to 50 student submissions. Further, the repository names only include the student's identifiers, not names, so that mix-ups could occur when importing the solutions into an IDE.

4.3 Discussion

While using Stager, we identified four main advantages: (1) Combining commits is particularly helpful to review all changes of one student at a glance. This allows the reviewer to immediately identify whether the student has understood the problem statement and has implemented a proper solution. (2) Renaming the projects simplifies the assessment and comparison of multiple solutions. The reviewer can import multiple solutions at the same time with one click into an IDE. It increases the confidence of the reviewers, so that the assessment is associated with the correct student. (3) While most students follow the deadline of an exercise, some students have committed changes after the deadline. It would be possible to remove write permissions for all student git repositories at the given deadline, but this might be hard to realize. Enforcing the deadlines in Stager is easier and filters the cases where students try to circumvent the deadline. (4) Stager only depends on using git repositories for programming exercises and other instructors can use it without adaptions in their courses, e.g. in GitHub Classroom or other git environments[17]. As Stager is open-source, other instructors can adapt it to their own needs.

While Stager is easy to use as a standalone tool, reviewers need to configure it for each exercise as described in Section 3.1. It would further simplify the

[17] https://classroom.github.com

Metric	1. Collision Detection	2. Serialization of Code	3. Adapter Pattern
Total submission count	1104	657	794
Total commit count	1998	3880	2447
Average amount of commits per student	1.81	5.91	3.08
Total commits after exercise deadline	34	8	7
Total submission count with at least one white space related change	125	118	183

Table 2: Quantitative analysis of submission metrics for three programming exercises of the course

configuration if Stager would be integrated into the exercise management system, where the instructor sets up the programming exercise. Then Stager would automatically know the submission deadline, the latest commit of the instructor in the code template, and the remote repository URL. This would make the use of Stager easier and seamlessly.

4.4 Limitations

Our experience report only included three exercises that used Stager for code reviews. It would be interesting to analyze the concrete time-savings with a comparison and to use Stager throughout the whole course. While we have first indications, we did not evaluate whether the quality of the reviews improved through the use of Stager.

In addition, Stager's implementation currently has the following limitations: (1) Reviewers have to manually search for each student repository's key the first time they use Stager, before being able to use Stager for the remaining steps. The previously mentioned integration of Stager into an exercise management system would overcome this step. (2) For every exercise, the *config.json* file has to be changed accordingly with the deadline, URL-schema, and commit of the instructor. This could also be adapted to be automatically included when creating exercises by means of an exercise management system. (3) Reviewers have to install Stager on their computer and start it via a double-click or the command line interface. A web-based solution or a plugin into an IDE (e.g. Eclipse) in which the reviewers import the code would provide a more user-friendly experience.

5 Conclusion

Manual code reviews are important for the learning experience of students. While automatic tests can find typical problems and check whether code works as intended, they cannot find all problems, code smells, and implementation issues. Automatic assessment imposes certain solutions on the students and might limit their creativity. Stager supports code reviewers by automating steps in the manual assessment of programming exercises to reduce effort for the preparation and the conduction of code reviews. Stager downloads multiple students' submissions, renames folders and projects, filters out late submissions, and fixes typical white space problems. All commits of one student are combined into one discrete change-set that is easier to review. Code reviewers can better distinguish between the submissions of multiple students and identify students' contributions more quickly.

Our experience in a course with 1600 students and 45 teaching assistants shows that Stager reduced the reviewing effort and time for teaching assistants. The reviewers used the saved time to write better reviews and give more detailed feedback to the students. This improved the student's learning. A quantitative analysis in three programming exercises shows that Stager identifies several late submissions and fixes many white space issues.

Stager is free, open source, and available under the MIT license, so that other instructors can use it in their courses[18]. We will continue the development and aim to integrate the tool into the automated assessment system ArTEMiS [Krusche and Seitz, 2018]. Our future work also includes the integration of code quality metrics to support the actual code assessment. This could make it easier for reviewers to spot code quality issues in the students' solutions and be included, e.g. as a text file, into the feedback pipeline.

In addition, we would like to evaluate the quality of the code reviews when using Stager compared to pure manual reviews with respect to the completeness, helpfulness, and understandability of the review. Depending on the results of this evaluation, we could integrate strategies to semi-automatically propose common code review feedback. Automatic suggestions would further reduce the effort of reviewers but allow them to tailor these suggestions to the concrete situation.

References

[Ala-Mutka 2005] ALA-MUTKA, Kirsti M.: A Survey of Automated Assessment Approaches for Programming Assignments. In: *Computer Science Education* 15, pages 83–102, 2005.

[Cerioli and Cinelli 2008] CERIOLI, Maura ; CINELLI, Pierpaolo: GRASP: Grading and Rating ASsistant Professor. In: *Proceedings of the Informatics Education Europe III Conference*, 2008.

[18] https://github.com/arubacao/stager

[Chen 2004] CHEN, P. M.: An automated feedback system for computer organization projects. In: *IEEE Transactions on Education* 47, pages 232–240, 2004.

[Gerdes et al. 2017] GERDES, Alex ; HEEREN, Bastiaan ; JEURING, Johan ; BINSBERGEN, L. T. van: Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback. In: *International Journal of Artificial Intelligence in Education* 27, pages 65–100, 2017.

[Heckman and King 2018] HECKMAN, Sarah ; KING, Jason: Developing Software Engineering Skills Using Real Tools for Automated Grading. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 794–799, 2018.

[Insa and Silva 2015] INSA, David ; SILVA, Josep: Semi-Automatic Assessment of Unrestrained Java Code: A Library, a DSL, and a Workbench to Assess Exams and Exercises. In: *Proceedings of the Conference on Innovation and Technology in Computer Science Education*, pages 39–44, 2015.

[Jackson 2000] JACKSON, David: A semi-automated approach to online assessment. In: *SIGCSE Bulletin* 32, pages 164–167, 2000.

[Jackson and Usher 1997] JACKSON, David ; USHER, Michelle: Grading Student Programs Using ASSYST. In: *Proceedings of the 28th Technical Symposium on Computer Science Education*, pages 335–339, 1997.

[Knobelsdorf and Romeike 2008] KNOBELSDORF, Maria ; ROMEIKE, Ralf: Creativity As a Pathway to Computer Science. In: *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, pages 286–290, 2008.

[Krusche et al. 2017a] KRUSCHE, Stephan ; BRUEGGE, Bernd ; CAMILLERI, Irina ; KRINKIN, Kirill ; SEITZ, Andreas ; WÖBKER, Cecil: Chaordic Learning: A Case Study. In: *Proceedings of the 39th International Conference on Software Engineering: Software Engineering Education and Training Track*, pages 87–96, IEEE, 2017.

[Krusche and Seitz 2018] KRUSCHE, Stephan ; SEITZ, Andreas: ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 284–289, 2018.

[Krusche et al. 2017b] KRUSCHE, Stephan ; SEITZ, Andreas ; BÖRSTLER, Jürgen ; BRUEGGE, Bernd: Interactive Learning: Increasing Student Participation through Shorter Exercise Cycles. In: *Proceedings of the 19th Australasian Computing Education Conference*, pages 17–26, 2017.

[Krusche et al. 2017c] KRUSCHE, Stephan ; VON FRANKENBERG, Nadine ; AFIFI, Sami: Experiences of a Software Engineering Course based on Interactive Learning. In: *Tagungsband des 15. Workshops "Software Engineering im Unterricht der Hochschulen"*, pages 32–40, 2017.

[Lawrance et al. 2013] LAWRANCE, Joseph ; JUNG, Seikyung ; WISEMAN, Charles: Git on the Cloud in the Classroom. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 639–644, 2013.

[McCracken et al. 2001] MCCRACKEN, Michael ; ALMSTRUM, Vicki ; DIAZ, Danny ; GUZDIAL, Mark ; HAGAN, Dianne ; KOLIKANT, Yifat Ben-David ; LAXER, Cary ; THOMAS, Lynda ; UTTING, Ian ; WILUSZ, Tadeusz: A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. In: *Working Group Reports on Innovation and Technology in Computer Science Education*, pages 125–180, 2001.

[Pieterse 2013] PIETERSE, Vreda: Automated Assessment of Programming Assignments. In: *Proceedings of the 3rd Computer Science Education Research Conference*, pages 45–56, 2013.

[Poženel et al. 2015] POŽENEL, Marko ; FÜRST, Luka ; MAHNIČ, Viljan: Introduction of the automated assessment of homework assignments in a university-level programming course. In: *38th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 761–766, IEEE, 2015.

[Robins et al. 2003] ROBINS, Anthony ; ROUNTREE, Janet ; ROUNTREE, Nathan: Learning and teaching programming: A review and discussion. In: *Computer Science Education* 13, pages 137–172, 2003.

[Staubitz et al. 2015] STAUBITZ, Thomas ; KLEMENT, Hauke ; RENZ, Jan ; TEUSNER, Ralf ; MEINEL, Christoph: Towards practical programming exercises and automated assessment in Massive Open Online Courses. In: *Teaching, Assessment, and Learning for Engineering*, pages 23–30, IEEE, 2015.

[Striewe and Goedicke 2013] STRIEWE, Michael ; GOEDICKE, Michael: Analyse von Programmieraufgaben durch Softwareproduktmetriken. In: *Tagungsband des 13. Workshops "Software Engineering im Unterricht der Hochschulen"*, pages 59–68, 2013.

Projektmanagement lernen ohne Projekt

Mario Winter, TH Köln
mario.winter@th-koeln.de

Zusammenfassung

Das Modul "Projektmanagement" (PM) wird seit dem Studienjahr 2003 an der Fakultät Informatik und Ingenieurwissenschaften der TH Köln (bis 09/2015 FH Köln) für alle Informatik-Studiengänge angeboten. Besondere Herausforderungen sind die seit 2012 zunehmend hohen Studierendenzahlen und die inhaltlichen und strukturellen Unterschiede der Curricula der vier beteiligten Bachelor-Studiengänge. Zudem lassen es unterschiedliche Gründe nicht zu, das PM unmittelbar auf ein reales (studentisches Lehr-) Software-Entwicklungsprojekt angewendet wird.

Dieser Beitrag beschreibt die Entwicklung des grundsätzlichen Lehr-/ Lern-Arrangements von PM hin zu einer kompetenzorientierten "Team-Teaching"-Veranstaltung für alle Studiengänge, den didaktischen Hintergrund und die Lehr-Erfahrungen sowie -Aufwände in der PM-Lehre mit über 350 Studierenden.

Abstract

Since the academic year 2003, the module "Project Management" (PM) has been offered at the Faculty of Computer Science and Engineering of the Technical University of Cologne (until 09/2015 FH Köln) for all computer science courses. Particular challenges are the increasing number of students since 2012 and the content and structural differences in the curricula of the four participating Bachelor programs. In addition, there are several reasons why PM is not applied directly to a real-world (student) software development project.

This article describes the development of the basic teaching / learning arrangement of PM towards a competence-oriented "Team Teaching" event for all study programs, the didactic background, and the teaching experience and efforts in PM teaching with more than 350 students.

Kurze Vorstellung

Ziel des Moduls "Projektmanagement" (PM, 5 ECTS CP) ist die Befähigung der Studierenden, die grundlegenden Aufgaben des PMs, insbesondere in IT-Projekten, zu charakterisieren und durchzuführen. Dabei sollen sie die PM-Methoden, -Techniken und -Werkzeuge zielgerichtet einsetzen und auch die erforderlichen soziologischen und kommunikativen Aspekte berücksichtigen können. Sie sollen mit dem Ziel einer menschengerechten und soziologisch fundierten Menschenführung eine wirkliche und optimale Produktivität bei komplexen Projekten erreicht können.

Besondere Herausforderungen sind die seit 2012 zunehmend hohen Studierendenzahlen und die inhaltlichen und strukturellen Unterschiede der Curricula der vier beteiligten Studiengänge (Informatik, Medieninformatik, Technische Informatik, IT Management und Wirtschaftsinformatik.

Die unterschiedliche Positionierung des Moduls in den Bachelor-Studiengängen – tw. im dritten, tw. im fünften Semester – lässt es nicht zu, das die Inhalte des Moduls von den Studierenden unmittelbar auf reale (studentische Lehr-) Software-Entwicklungsprojekte angewendet werden. Dies erscheint auch aufgrund der großen Abhängigkeiten zwischen Projektmanagement, Erfahrung in der Softwareentwicklung und Projekterfolg als zu riskant.

Der Beitrag beschreibt die Entwicklung des grundsätzlichen Lehr-/ Lern-Arrangements, also der Lehr-, Hausarbeits- und Prüfungsformate des Moduls Projektmanagement von eher inhaltsorientierten "Single-Teacher"-Veranstaltungen pro Studiengang hin zu einer gemeinsamen kompetenzorientierten "Team-Teaching"-Veranstaltung für alle Studiengänge, den didaktischen Hintergrund und die Lehr -Erfahrungen sowie -Aufwände in der Projektmanagement-Lehre mit über 350 Studierenden.

Historie und Inhalte

Im Sommersemester 2003 wurde PM im 6. Semester des damaligen Diplom-Studiengangs „Allgemeine Informatik“ der FH Köln erstmals vom Autor und einer Mitarbeiterin[1] durchgeführt. Die Veranstaltung war im damaligen Curriculum mit 2 SWS Vorlesung und 2 SWS Praktikum verankert. Teilgenommen haben 53 Studierende.

[1] Im Folgenden meint „wir“ die an PM beteiligten Kollegen und Mitarbeiterinnen sowie Mitarbeiter.

Einführung und Überblick, *1. Vorlesung: 20.03.2003*

Organisatorische Aspekte der Vorlesung und insbesondere des Praktikums (Einteilung der Studierenden in-Teams). Danach werden Definitionen zentraler Begriffe wie „(Software-) Projekt“ und „Management“ erarbeitet und ein Überblick über die Aktivitäten beim Management von Softwareprojekten – und damit den weiteren Inhalt der Vorlesung – gegeben.

Die „menschliche Komponente“ *2. Vorlesung: 27.03.2003*

Gegenstand dieser Vorlesung sind die – nicht nur für Softwareprojekte – zentralen Fragen der Team Bildung, der Zusammenarbeit im Team und der Motivation von Kollegen und Mitarbeitern. Grundlegende für diese Fragen ist, die Individualität der Menschen anzuerkennen, ihre Stärken zu nutzen und mit den Schwächen umzugehen. Da hierzu die “richtige“ Kommunikation wesentlich ist, werden zunächst Kommunikationsmodelle und -muster vorgestellt. Darauf – und auf dem Phänomen “Motivation“ – aufbauend werden typische Merkmale der Gruppenbildung erläutert und Hinweise zum Krisenmanagement gegeben.

Kosten/Nutzen-Analysen und Entscheidungstechniken*3. Vorlesung: 03.04.2003*

Diese Vorlesung betrachtet Kosten/Nutzen-Analysen und Entscheidungstechniken für Softwareprojekte. Den Ausgangspunkt bildet eine Klassifikation der Kosten- und der Nutzenarten. Ein besonderes Augenmerk liegt hierbei auf der Differenzierung in quantifizierbare und nicht quantifizierbare Kosten- und insbesondere Nutzenarten sowie auf dem Aspekt der Unsicherheit der Plandaten. Anschließend werden statische Verfahren der Investitionsrechnung auf Investitionen in Software angewendet. Zur Einbeziehung nicht-monetärer Entscheidungskriterien runden eine Einführung in ein Entscheidungsmodell sowie Entscheidungstechniken wie z.B. die Nutzwertanalyse unter Sicherheit, Risiko und Unsicherheit diese Vorlesung ab.

Organisation *4. Vorlesung: 10.04.2003 Aufbauorganisation, 5. Vorlesung: 17.04.2003 Ablauforganisation*

Bei der Projektorganisation geht es zum einen um die Aufbauorganisation, bei der die Integration des Projekts in das Unternehmen und die projektinterne Aufbauorganisation unterschieden wird. Andererseits wird die Ablauforganisation im Rahmen von Softwareprojekten in der Regel durch Vorgehensmodelle beschrieben. Wir skizzieren “klassische“ Vorgehensmodelle, bei denen die Aktivitäten in strenger Reihenfolge jeweils vollständig durchgeführt werden, als auch modernere Vertreter, die iterativ vorgehen und das Produkt in kleinen Schritten – sozusagen inkrementell – entwickeln.

Aufwandsschätzung *6. Vorlesung: 08.05.2003 Projektumfang 7. Vorlesung: 15.05.2003 Entwicklungszeit*

Als Grundlage der Projekt-Durchführungsentscheidung sowie -Planung betrachten wir in diesen Vorlesungen Verfahren zur Aufwandsschätzung für Software-Entwicklungsprojekte. Ziel ist zum einen, konkrete Zahlen zu liefern und zum anderen, eine fundierte Ressourcenplanung für das Projekt zu ermöglichen. Nach einfachen Verfahren wie der Prozentsatzmethode gehen wir auf die Function-Point-Analyse zur Schätzung der Produktkomplexität anhand der im Lastenheft skizzierten Produktanforderungen ein. Es folgt das COCOMO-Verfahren zur Abschätzung der Entwicklungszeit anhand des - vorher zu schätzenden – Produktumfangs.

Planung und Risikomanagement *8. Vorlesung: 22.05.2003 Projektplanung I 9. Vorlesung: 05.06.2003 Projektplanung II 10. Vorlesung: 12.06.2003 Risikomanagement*

Inhalt dieser Vorlesungen sind die detaillierte Projektplanung und das Risikomanagement für Softwareprojekte. Für die Projektplanung werden u.A. die Verfahren der Netzplantechnik und ihre Anwendung in einem Software-Entwicklungsprojekt behandelt. Grundlage ist die vorgangsorientierte Zeit- und Ressourcenplanung. Risiken verbleiben auch bei bester Planung - im Rahmen des Risikomanagements versucht man, die wichtigsten Risiken zu identifizieren, zu priorisieren und Möglichkeiten zu ihrer Beherrschung zu finden.

Controlling und Abschluss *11. Vorlesung: 26.06.2003 Projektcontrolling und Projektabschluss*

Im Hinblick auf die Steuerung und das Controlling von Softwareprojekten besprechen wir Möglichkeiten, die Zeit- und Ressourcenplanung auf aktuelle Planabweichungen anzupassen. Dabei betrachten wir auch den Einsatz von Kennzahlen und das Berichtswesen. Im letzten Teil befassen wir uns mit dem für die „lernende Organisation“ wichtigen Projektabschluss.

Querschnittsthemen *12. Vorlesung: 03.07.2003*

Diese Vorlesung widmet sich den ergänzenden Bereichen Dokumentation sowie Versions- und Konfigurationsmanagement. Einige Aspekte der Mitarbeiterführung wie die Einstellung neuer Projektmitarbeiter sowie die Personalentwicklung runden die Vorlesung ab.

Abb. 1 PM-Inhalte (Auszüge aus dem Skript des Autors von 2003)

Die Inhalte der Vorlesungen orientierten sich an verbreiteten Lehrbüchern wie z.B. (Feyhl & Feyhl 1996), (Balzert 1998), (Grupp 1998) und (Friedlein 2003) sowie den Lerneinheiten zum Fernstudienkurs „Management von Softwareprojekten" der FernUniversität Hagen (Henrich, 2001). Im Praktikum wurden diese an einem vorgegebenen Fallbeispiel angewendet (s.U.). Im Laufe der Jahre wurden immer wieder aktuellere Werke wie z.B. (Aichele, 2014), (Broy & Kuhrmann, 2013) und (Kerzner, 2009) einbezogen.

In Abb. 1 sind die in den wöchentlichen Frontalvorlesungen behandelten Themenbereiche skizziert (Auszüge aus dem Skript des Autors von 2003).

Bereits im Sommersemester 2004 wurde die Veranstaltung von zwei weiteren Lehrenden parallel auch in den beiden Studiengängen „Medieninformatik" und „Technische Informatik" durchgeführt. Aus den drei beteiligten Studiengängen nahmen insgesamt 164 Studierende teil.

Ab 2006 waren alle Studiengänge auf das Bachelor-Modell umgestellt. Die Straffung der Curricula auf 6 Semester führte auch zu einer Vereinheitlichung der Studienverlaufspläne, mit einem fast identischen „Stamm" in den ersten beiden Semestern und spezialisierenden „Ästen" in den Semestern 3-6. PM war nun in allen Bachelor-Curricula im 5. Semester mit 5 ECTS-Punkten angesiedelt, nach wie vor mit 2 SWS Vorlesung und 2 SWS Praktikum.

Mit der ersten Reakkreditierung der Studiengänge im Jahr 2012 erfolgte eine Differenzierung der Studienverlaufspläne: PM blieb in drei Bachelor-Curricula im 5. Semester, in einem (Wirtschaftsinformatik) jedoch nun im 3. Semester. Der studentische Aufwand betrug weiterhin 5 ECTS-Punkte, wie zuvor mit 2 SWS Vorlesung und 2 SWS Praktikum.

Studierendenzahlen

Nachdem im Sommer 2004 noch 164 Studierende aus den vier Studiengängen teilgenommen haben, nahm deren Zahl in den darauffolgenden Jahren tw. aufgrund der Erhebung von Studiengebühren in NRW bis 2009 tendenziell eher ab. Beginnend mit dem Studienjahr 2010 führten die Aussetzung der allg. Wehrpflicht zum 1. Juli 2011, die Abschaffung der Studiengebühren in NRW zum Wintersemester 2011/12 sowie der „Doppel-Abiturjahrgang" im Jahr 2013 (inkl. Hochschulpakt NRW) zu einem stark ansteigenden Trend der Zahlen, der sich seitdem fortsetzt und nun um ein Plateau von ca. 300 Studierenden pendelt. Spitzenreiter war das Wintersemester 2017/18 mit 359 Studierenden.

Die Teilnehmerzahlen seit Wintersemester 2010/2011 zeigt die Tabelle 1.

Semester	Studierende
2010/11	143
2011/12	102
2012/13	150
2013/14	182
2014/15	303
2015/16	261
2016/17	263
2017/18	359
2018/19	286

Tabelle 1 PM Teilnehmerzahlen seit 2010

Entwicklung des Lehr-/Lernarrangements

Entwicklungsphase I: Inhaltsorientierung und Single-Teacher-Setting

Die erste Durchführung von PM im Sommersemester 2003 für den Diplom-Studiengang „Allgemeine Informatik" erfolgte durch den Autor im "Single-Teacher"- Lehr-/ Lern-Arrangement. Über die gesamte Vorlesungszeit wurden die Inhalte der in Abb. 1 skizzierten Frontalvorlesungen im Praktikum in praktischen Übungen vertieft und in einer Klausur „abgeprüft".

Dazu bildeten die Studierenden Teams mit 6 Team-Mitgliedern, in denen Sie die Übungsaufgaben bearbeiteten. Jede Aufgabe erforderte die Anwendung einer bestimmten Technik auf einen vorgegebenen Projektgegenstand. Grundlage bildete ein vom Autor erstelltes Lastenheft für eine Anwendung zur Seminarverwaltung, angelehnt an das bekannte Fallbeispiel aus (Balzert, 1996). Durch eine inhaltliche Verzahnung der Aufgaben über den Projektgegenstand wurde angestrebt, die Planungs- und Überwachungstätigkeiten im Projektmanagement realitätsnah abzubilden.

Abnahmen der Lösungen zu den Übungsaufgaben inklusive formativem Feedback der Projektergebnisse erfolgten im 2-wöchigen Rhythmus durch den Autor und eine Mitarbeiterin.

Das summative Feedback – also die Modulprüfung – erfolgte in Form einer 120-minütigen Klausur. Neben einigen Wissensfragen mussten die Studierenden 3 bis 4 „konstruktive“ Aufgaben z.B. zur Nutzwert-Analyse, Aufwandschätzung mit Function-Point-Analyse oder Netzplanung bearbeiten (s. Abb. 2).

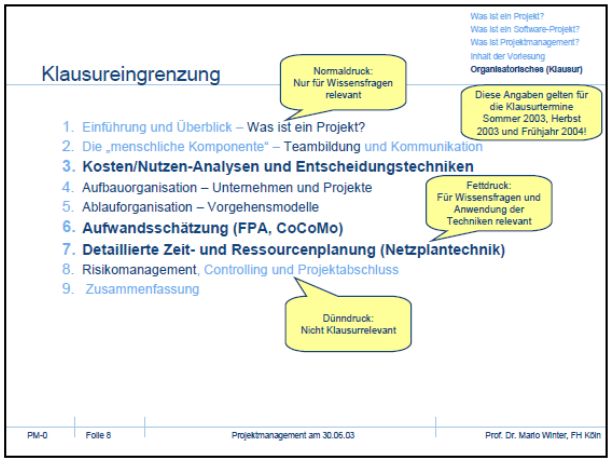


Abb. 2 Klausurthemen 2003

Entwicklungsphase II: Realitätsnähe und Team-Teaching

Unsere Diskussionen vor der ersten parallelen Durchführung von PM ab dem Sommersemester 2004 in den vier Studiengängen „Allgemeine Informatik“, „Medieninformatik“, „Technische Informatik“ und „Wirtschaftsinformatik“ basierten i.W. auf den Eindrücken während der Praktikum-Abnahmen und den Ergebnissen der ersten Klausuren. Wir identifizierten folgende Probleme des initialen Lehr-/Lernformates:

- Die Studierenden bearbeiteten die Praktikum-Aufgaben nicht wirklich als Team, sondern gemäß „Teile und Herrsche“ eher in Einzelarbeit. Dadurch wurden Querbezüge in den Aufgabenstellungen und der Anwendung der Methoden bzw. Techniken oft nicht erkannt, worunter die Konsistenz der Lösungen erheblich litt.
- Zusätzlich waren viele Studierende „Experten“ genau für die PM-Methode bzw. PM-Technik, welche sie selbst im Praktikum aktiv angewendet haben, zeigten aber tw. erhebliche Lücken in anderen Bereichen.
- Auch wurde vielen Studierenden nicht bewusst, dass die Bearbeitung jeder einzelnen Aufgabe selbst ein „Mini-Meta-Projekt“ darstellt. „Meta“ in dem Sinne, dass dabei Arbeitspakete für das PM von Arbeitspaketen für den vorgegebenen Projektgegenstand geplant und auch durchgeführt und gesteuert werden müssen (während die Arbeitspakete für den vorgegebenen Projektgegenstand in PM nur geplant, aber nicht gesteuert und durchgeführt werden müssen).
- Last but not least waren viele Studierende nicht wirklich motiviert dabei, da die vorgegebene Aufgabenstellung (Anwendung zur Seminarverwaltung, s.O.) eher angestaubt, akademisch und fachlich sowie technisch uninteressant erschien.

Ein erstes Brainstorming konzentrierte sich auf den Projektgegenstand. Schnell kam die Frage auf, warum wir in PM nicht ein wirkliches Studierendenprojekt planen und dann tatsächlich auch durchführen lassen? Dagegen sprach, dass im Rahmen der 5 ects cp zusätzlich zum Erlernen von PM kein noch so bescheidenes „Entwicklungsprojekt“ möglich ist. Eine Verknüpfung von PM mit den im Curriculum verankerten Studien- und Praxisprojekten war nicht in der Breite durchführbar, da viele solcher Projekte in Zusammenarbeit mit externen Projektpartnern durchgeführt werden und eben fundierte PM-Kenntnisse bereits voraussetzen. Das Risiko, dass „Fehlplanungen“ den Projekterfolg gefährden, war uns einfach zu groß[2].

Beschlossen wurde, die Studierenden-Teams immerhin einen eigenen Projektgegenstand im Bereich Softwareentwicklung definieren zu lassen, um damit die intrinsische Motivation zu fördern.

Diese und einige weitere Diskussionen zum „Team-Setting“ führten schließlich zu folgenden Veränderungen der Praktika:

1. Die Teams wurden heterogen mit 6 Studierenden aus mindestens 3 Studiengängen gebildet. Dies sollte die „Diversität“ in der Besetzung realer Projektteams widerspiegeln.
2. Es sind 6 Projektaufgaben zu bearbeiten:
   I) Erstellung des Lastenheftes;
   II) Aufwandsschätzung;
   III) Kosten-/Nutzen-Betrachtung;
   IV) Risikomanagement;
   V) Vorgehensmodell-Tailoring;
   VI) Zeit- und Ressourcenplanung.
3. Für jede dieser 6 Projektaufgaben musste das Team eine Rollenzuteilung festlegen:
   - Ein Projektleiter (PL);
   - Ein Repräsentant (PR);
   - 2-4 Mitarbeiter (Bearbeiter, Rechercheur, Reviewer, …).

[2] Dies haben u.A. auch Fleischmann und Spies an den TU's Darmstadt und München festgestellt. Sie schreiben dazu, *„dass die größten Herausforderungen für die beteiligten Studierenden dabei weniger in der Bewältigung technischer oder fachlicher Fragestellungen liegen: Diese Problematik ist ihnen aufgrund ihrer Lernerfahrung im Studienverlauf bereits geläufig. Vielmehr stellt der Umgang mit organisatorischen und sozialen Problemen eine viel größere Herausforderung dar und führt daher oft zu für die Studierenden unvorhergesehenen Problemen im Projekt.“* (Fleischmann & Spies, 2005, S. 26)

Damit sollte die Bearbeitung der Aufgaben als „Mini-Meta-Projekt“ im Team gefördert werden.

4. Jedes Team definierte einen eigenen Projektgegenstand gemäß der groben Vorgabe, dass dieser innerhalb vorgegebener Budget- (500.000 € bis 1.000.000 €) und Aufwands- bzw. Zeit-Grenzen (36 bis 72 PM, 6 bis 12 Monate) liegt. Dies sollte einerseits die Motivation der Studierenden steigern und andererseits den „Ergebnisaustausch“ zwischen den Teams mindern.

In einem Team-Teaching-Setting mit drei Lehr-Tandems aus Kollege und Mitarbeiterin bzw. Mitarbeiter wurden die Vorlesungen dann zwar nach wie vor über die gesamte Vorlesungszeit, jetzt aber im Wechsel von den drei beteiligten Kollegen gehalten. In dieser Zeit wurde die Bearbeitung der Aufgaben im Praktikum je Team von einem Kollegen und einer Mitarbeiterin oder einem Mitarbeiter betreut.

Die Bearbeitung der Praktika bestand aus zwei Teilen:

1. Jedes Team erstellte jeweils unter der Projektleitung eines/einer Studierenden die Artefakte inkl. der Begründung der Vorgehensweise und Ergebnisse zu jeder der Aufgaben I) bis VI);
2. In einem Abschluss-Kolloquium präsentierten die Team-Mitglieder nacheinander in jeweils 5 Minuten die Ergebnisse einer der sechs Aufgaben, wobei einige Verständnisfragen gestellt wurden.

Die selbstdefinierten Projektgegenstände waren in der Regel reine Softwareentwicklungen (z.B. Parkplatz-Reservierung im Web, Client-/Server-Anwendung zur Raumreservierung) und nur einige wenige HW/SW-Systementwicklungen. Die Motivation der Studierenden stieg durch den selbstgewählten Projektgegenstand merklich, wobei die Präzisierung der Projektvision durch das Lastenheft für viele „Aha-Momente“ bzgl. unterschiedlicher Interpretationen der Projektidee sorgte. Die Modulnote ergab sich weiterhin alleine aus einer Klausur.

Ein wesentlicher Punkt des eingeholten Feedbacks der Studierenden war der Wunsch, die teambasierten Leistungen bei der Bearbeitung der Projektaufgaben auch in die summative Bewertung, also die Modulnote einließen zu lassen.

Entwicklungsphase III: Lernziel-Orientierung und veranstaltungsbegleitende Prüfungsformate

Im Rahmen der Umstellung auf das Bachelor-Modell wurde erstmals eine Modulbeschreibung mit den Lernzielen bzw. den „von den Studierenden zu erlangenden Kompetenzen“ gefordert. Das von uns formulierte Modulziel zeigt Abb. 3.

Die Studierenden sollen befähigt werden,

- die grundlegenden Aufgaben des Projektmanagements, insb. in IT-Projekten, zu charakterisieren und durchzuführen;
- die Projektmanagement-Methoden, -Techniken und -Werkzeuge zielgerichtet einzusetzen;
- die erforderlichen soziologischen und kommunikativen Aspekte zu berücksichtigen, insb. mit dem Ziel einer menschengerechten und soziologisch fundierten Menschenführung zur Erreichung einer wirklichen und optimalen Produktivität bei komplexen Projekten.

Abb. 3 Modulziel von PM 2006

Ende 2006 waren dann alle Informatik-Studiengänge vom Diplom auf den Bachelor umgestellt. Das oben genannte Feedback der Studierenden, die Verständigung auf das explizit formulierte Lernziel sowie unsere zunehmende Erfahrung mit der Einschätzung und Bewertung der von den Studierenden im Team erzielten Praktikum Ergebnisse und Präsentationen ermutigten uns dann ab dem Sommersemester 2007, die Modulnote anhand der Teamleistung und der individuellen Leistung (als PL und PR) im Praktikum und im Abschluss-Kolloquium zu ermitteln. (Inhaltlich wurde 2007 das bis dahin für die Ablaufplanung verwendete V-Modell 97 durch den Nachfolger V-Modell XT ersetzt.)

Zur besseren Einschätzung der Teamleistung musste nun jedes Team zu den Aufgaben neben den Bearbeitungsergebnissen auch Protokolle abgegeben. Diese sollten das Zustandekommen der jeweiligen Ergebnisse im Vorfeld der Präsentation dokumentieren, insbesondere Informationen zur Team-/Rollenaufteilung, zur Bearbeitung der Aufgaben (evtl. mit den jeweiligen Teilschritten) und zum jeweiligen Arbeitsfortschritt. Auch für ein angemessenes Layout der Protokolle war zu sorgen.

Die gesamte Bewertung der einzelnen Studierenden ergab sich zum einen aus der Bewertung der Dokumentation zu der als PL bearbeiteten Aufgabe (Hausarbeit). Es wurden jedoch im Verlaufe des Praktikums durch die Präsentationen, die Protokolle und in den Fragerunden „Punkte“ gesammelt, die sich positiv in der Gesamtbewertung niederschlugen.

Insgesamt wählten wir folgende Gewichtung der einzelnen Prüfungsbestandteile:

- Dokument (Hausarbeit) 40 Punkte.
- Präsentation und Fragen 40 Punkte
- Protokolle/Teamleistung 20 Punkte

Die Bewertung der Dokumente erfolgte separat für jedes Team alleine durch den Kollegen im betreuenden Lehr-Tandem. Dies bedeutete für jeden von uns drei Kollegen die Bewertung der im Praktikum er-

stellten Arbeitsergebnisse, begründenden Dokumente sowie Protokollen von ca. 10 Praktikum-Teams.

Die Präsentationen der Studierenden fanden geblockt an drei Tagen am Ende der Vorlesungszeit statt und wurden pro Team separat vom Kollegen und der Mitarbeiterin bzw. dem Mitarbeiter des betreuenden Lehr-Tandems bewertet.

Zum Schluss wurden die Bewertungen zusammengeführt und ergaben die Modulnote. Die hierbei unvermeidlichen Unterschiede in den subjektiven Einschätzungen wurden durch das in Tabelle 2 gezeigte Gesamt-Bewertungsschema für die Projektergebnisse sowie eine grobe Kriterienliste für die Präsentationen und Fragen (Tabelle 3) abgemildert. Bei der Bepunktung der Protokolle wurde das in den Protokollen beschriebene Vorgehen des Teams inkl. Teilaufgaben-Zuteilung, Zeitplanung und „Konfliktlösungen“ gewertet.

In diesem Lehr-/Lernarrangement wurde die Veranstaltung dann mehrfach durchgeführt. Das Feedback der Studierenden war durchaus positiv, allerdings wurden insbesondere unsere Mitarbeiterinnen und Mitarbeiter von einzelnen Studierenden „hinter vorgehaltener Hand“ auf die teilweise eher „gefühlte“, tw. aber auch nachvollziehbare Diskrepanz der Benotungen durch die drei Lehr-Tandems hingewiesen.

PROJEKTMANAGEMENT SS 2007										Nur die gelben Felder ausfüllen!											Durchschnitt:	1,60	
																					Anzahl:	24,00	
IDENTIFIKATION TEILNEHMER UND GRUPPE						AUSARBEITUNG				PRÄSENTATION						TEAM							
			GEWICHTUNG -->				0,7	0,3	0,4		0,4	0,3	0,1	0,2	0,4	0,334	0,333	0,333	0,5	0,5	0,2		
				AI/MI/TI		PL bei		Form/	Note	Repr. bei	Inhalt/				Note	Protokolle Aufgaben			Proto-	Team	Note	NOTE	NOTE
Nr.	Name	Vorname	Matr.Nr.	Team	Gruppe	Aufgabe	Inhalt	Darst.	Ausarb.	Aufgabe	Darst.	Stil	Zeit	Fragen	Präs.	1+2	3+4	5+6	kolle	Präs.	Team	GESAMT	gegeben
1				MI	1	1	1,3	1,5	1,4	3	1,5	1,3	1,0	1,7	1,4	1,7	1,5	1,7	1,6	1,3	1,5	1,41	1,3
2				MI	1	2	1,5	1,5	1,5	6	1,3	1,5	1,7	1,5	1,4						1,5	1,47	1,3
3				MI	1	3	1,3	1,5	1,4	2	1,5	1,5	1,0	1,5	1,5						1,5	1,42	1,3
4				MI	1	4	1,0	1,5	1,2	1	1,5	1,7	1,0	1,7	1,6						1,5	1,37	1,3
5				MI	1	5	1,0	1,5	1,2	4	1,5	1,7	2,3	1,7	1,7						1,5	1,43	1,3
6				MI	1	6	1,3	1,5	1,4	5	1,5	1,7	2,0	1,5	1,6						1,5	1,48	1,3
7				TI	2	1	1,3	1,5	1,4	5	1,7	1,7	1,0	2,0	1,7	1,5	1,5	1,7	1,6	1,3	1,4	1,51	1,3
8				MI	2	2	1,5	1,5	1,5	1	1,5	1,3	1,0	1,0	1,3						1,4	1,40	1,3
9				MI	2	3	1,3	1,5	1,4	4	1,7	2,0	1,0	1,5	1,7						1,4	1,50	1,3
10				MI	2	4	1,3	1,3	1,3	2	2,0	1,5	1,0	2,3	1,8						1,4	1,53	1,3
11				MI	2	5	1,7	1,5	1,6	6	1,3	1,3	1,0	1,7	1,4						1,4	1,48	1,3
12				MI	2	6	1,7	1,5	1,6	3	1,7	2,0	1,0	1,7	1,7						1,4	1,63	1,7
13				MI	3	1	1,5	1,3	1,4	3	1,7	2,0	1,3	1,7	1,8	1,5	1,7	1,7	1,6	1,4	1,5	1,58	1,3
14				MI	3	2	1,7	1,3	1,6	6	1,7	1,7	1,7	2,0	1,8						1,5	1,64	1,7
15				MI	3	3	1,7	1,3	1,6	1	1,5	2,0	2,0	1,5	1,7						1,5	1,62	1,7
16				MI	3	4	1,3	1,3	1,3	2	1,5	2,0	1,7	1,5	1,7						1,5	1,49	1,3

Tabelle 2 Gesamt-Bewertungsschema SoSe 2007

Fragen	Stil
A____F____: _____ A____F____: _____ A____F____: _____ A____F____: _____ A____F____: _____ A____F____: _____ A____F____: _____	Rede: Sicher / Flüssig / Stockend / Zum Zuhörer / Unterhaltsam Inhalt: Vollständig + Richtig / Sachkundig / Überlegt / tw. falsch Sprache: Fachsprachlich / Teils-Teils / Umgangssprachlich Dialog: Geht auf Fragen ein / Weicht aus / Chaotisch

Tabelle 3 Bewertungsschema Präsentation und Fragen SoSe 2007

Im Rahmen der 2009 beginnenden Vorbereitungen der Reakkreditierung der Studiengänge stand zunächst die Lernzielorientierung der Curricula im Vordergrund. Insbesondere hatten wir Probleme, mit den „herkömmlichen“, eher inhaltsorientierten Beschreibungsmitteln begründet die übergeordneten Lernziele (learning outcomes, vgl. Thurner & Böttcher et al. 2015) der Studiengänge auf deren Curricula herunter zu brechen bzw. deren Erfüllung (oder Erfüllbarkeit) aus den Lernzielen der einzelnen Module abzuleiten. Dies erforderte eine erneute Fokussierung auf die für PM formulierten Lernziele.

Wir präzisierten zunächst unsere bis dahin nur implizit im Lehr-/Lern-Arrangement vorhandenen Ansätze zur Erreichung der Lernziele von PM. Schnell wurde klar, dass die bislang von den Lehr-Tandems unterschiedliche Handhabung der Veranstaltungs-Bestandteile vereinheitlicht und den Studierenden explizit zu Beginn des Semesters verdeutlicht werden muss.

Die erste Veränderung bestand somit in einer expliziten Formulierung der Modalitäten für die veranstaltungsbegleitenden Prüfungsformate in Form

eines „Lehr-/Lern-Vertrages“ (Abb. 4). Das (positive) Feedback der Studierenden hierzu ergab sich implizit dadurch, dass die Abgabe-Meilensteine zu den Projektaufgaben von deutlich weniger Projektleitungen „gerissen“ wurde wie in den früheren Semestern.

Die zweite Veränderung betraf die Bewertung der Dokumente aus den Aufgabenlösungen der Teams. Hier versuchten wir, durch unterschiedliche Zuteilungen der Aufgaben zu den Lehr-Tandems zu einer gleichen Bewertung aller Teams zu gelangen.

Zunächst erstellten wir zur möglichst objektiven Bewertung der Aufgabenlösungen feingranulare „Controlling“ Bewertungsformulare. Dann teilten wir die Aufgabenlösungen aller Teams abwechselnd jeweils einem den Lehr-Tandems zu, so dass z.B. Lehr-Tandem A die Aufgaben 1 und 3 bewertete. Die „Lücke“ zwischen den beiden zu bewertenden Aufgabenstellungen und die bereits genannten Abhängigkeiten erforderten jedoch bei der Bewertung der zweiten Aufgabe auch eine nähere Ansicht der Lösung zur unmittelbar davorliegenden Aufgabe.

In den beiden darauffolgenden Semestern bewertete daher jedes Lehr-Tandem zwei unmittelbar aufeinanderfolgende Aufgaben. Daraus ergab sich eine deutlichere, jeweils nach einem Drittel der Vorlesungszeit geblockte „Prüfungsbelastung“. Auch bei dieser Aufteilung war für die beiden Lehr-Teams zu den Aufgaben 3 und 4 bzw. 5 und 6 ein Einblick in die davorliegenden Aufgabenlösungen der Teams notwendig.

Die dritte Veränderung führte zu einer Entzerrung der Präsentationstermine, die nun nicht mehr geblockt am Ende der Vorlesungszeit, sondern geblockt nur noch für jeweils zwei Aufgaben nach der Meilenstein-Abgabe zu den Projektaufgaben erfolgte. Dadurch wurde eine zeitnahe „Prüfung“ der erreichten Kenntnisse möglich, allerdings auf Kosten einer gewissen „Ungleichbehandlung“ der Studierenden, da ja die Repräsentanten der ersten beiden Aufgaben nicht auf Erfahrungen aus vorherigen Präsentationsterminen aufbauen konnten.

Hausarbeit (Dokumentation der Ergebnisse einer Aufgabe)

Zu den 6 Aufgaben muss das Team eine Dokumentation (Hausarbeit) erstellen. Für die Erstellung der Dokumentation ist dasjenige Teammitglied verantwortlich, das für die jeweilige Aufgabe die Rolle Projektleiter ausfüllt. Es muss in der Rolle des Projektleiters dafür sorgen, dass das Team ihm bei der Erstellung der Dokumentation tatkräftig zur Seite steht und die richtigen Teilaufgaben jeweils an die anderen Teammitglieder verteilen. In diesem Sinne ist jeder aus dem Team für eine Aufgabe als „Projektleiter“ verantwortlich und muss deren Ergebnisse in einer Hausarbeit dokumentieren.

Auf dem Weg bis zur endgültigen Abgabe der Hausarbeiten müssen die Ergebnisse in insgesamt 3 Abnahmeterminen präsentiert werden. Hierzu wird jedes Teammitglied die Ergebnisse einer Aufgabe bei der er/sie nicht die Rolle Projektleiter spielt, in der Rolle als Repräsentant präsentieren. Die 6 Dokumentationen werden dann zum 3. Abnahmetermin abgegeben.

Abnahmen der Praktika

Die Projektergebnisse werden zu je zwei Aufgaben in insgesamt 3 Abnahmeterminen (siehe Terminplan) vor den verantwortlichen Professoren präsentiert und das gesamte Team muss sich einer Reihe von Fragen hierzu stellen.

Präsentation der Ergebnisse durch Repräsentanten

In den Abnahmeterminen sind die Ergebnisse zu den beiden jeweils betroffenen Aufgaben zu präsentieren. Bei der Präsentation zu einer Aufgabe stellt ein Repräsentant (nicht der Projektleiter selber) die Ergebnisse zur Aufgabe anhand einiger Folien in genau 5 Minuten vor.

Fragerunde

Nach der Präsentation zu den beiden Aufgaben wird in einer ca. 20 minütigen Fragerunde das gesamte Team mit einer Reihe von Fragen zu den beiden Aufgaben konfrontiert und die einzeln befragten Teammitglieder müssen diese beantworten.

Protokolle

Zusätzlich müssen vom Team zu den beiden in der Abnahme zu präsentierenden Aufgaben und zu den zugehörigen Einstimmungsaufgaben Protokolle abgegeben werden, die das Zustandekommen der jeweiligen Ergebnisse im Vorfeld der Präsentation dokumentieren. Die Protokolle enthalten insbesondere Informationen zur Team-/Rollenaufteilung, zur Bearbeitung der Aufgaben (evtl. mit den jeweiligen Teilschritten) und dokumentieren den jeweiligen Arbeitsfortschritt. Auch für ein angemessenes Layout der Protokolle ist zu sorgen.

Abgabe der Hausarbeiten eine Woche nach der 3. Abnahme

Die dokumentierten Endergebnisse zu allen 6 Aufgaben werden dem jeweiligen Prüfer eine Woche nach dem 3. Abnahmetermin als ausgedrucktes finales Dokument übergeben.

Abb. 4 Explizite Veranstaltungsmodalitäten ab WiSe 2010/11

Entwicklungsphase IV: Kompetenzorientierung

Im Rahmen seiner hochschuldidaktischen Weiterbildung zum Fakultäts- Multiplikator für kompetenzorientierte Prüfungen im Sommer 2014 konnte der Autor die bisherigen Ansätze und Erfahrungen im Modul PM reflektieren.

Die Kompetenzorientierung ist ein wichtiges Mittel, Lehre und Prüfung voneinander zu trennen und gleichzeitig sowohl Lehre als auch Prüfungen zielgerichtet, effektiv, transparent, valide und zuverlässig zu gestalten. Abb. 5 zeigt den damit entstehenden Dreiklang von Lernziel, Lehr-/Lern-Arrangement und Prüfungsform(en) nach (Biggs & Tang, 2011).

Insbesondere bei Lernzielen auf höheren kognitiven Stufen, wie sie ja in PM erreicht werden sollen, bietet die Kompetenzorientierung einen begründeten Handlungsrahmen, in dem sich auch genügend Spielraum für die individuelle Ausgestaltung findet. Für projektorientierte SE-Lehrveranstaltungen zeigen dies z.B. (Böttcher & Thurner 2011 und Hummel, 2013).

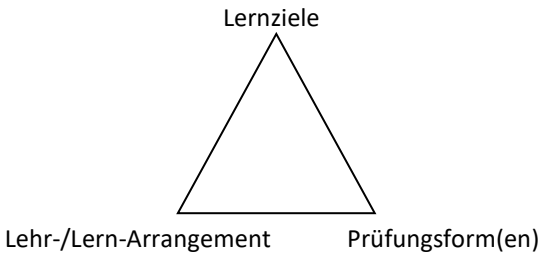


Abb. 5 Constructive Alignment

Gegen die Kompetenzorientierung wird oft dahingehend argumentiert, dass es i.S. der Freiheit von Forschung und Lehre nicht in erster Linie darum gehen kann, „berufsfähige" Absolventen zu „produzieren". Diese Argumentation greift m.E. zu kurz, da hierbei der Begriff Kompetenz i.d.R. nur i.S.d. Berufsbildungsforschung gesehen wird[2]. Kurz gefasst meint Kompetenz hier genau solche Fähigkeiten und Fertigkeiten, die in der heutigen industriellen bzw. beruflichen Praxis gefordert werden.

Darüber hinaus werden oft kommunikative und reflexive Fähigkeiten sowie „höhere" Fähigkeiten wie evaluieren und synthetisieren nicht betrachtet. Hierzu sollte man sich §2 (1) des HRG vor Augen halten: „Die Hochschulen […] bereiten auf berufliche Tätigkeiten vor, die die Anwendung wissenschaftlicher Erkenntnisse und wissenschaftlicher Methoden oder die Fähigkeit zu künstlerischer Gestaltung erfordern." Grundlage der Kompetenzorientierung in der Hochschule sollte der akademische Kompetenzbegriff sein, welcher folgende Punkte umfasst (Reis 2010):

- Die wissenschaftliche Modellierung komplexer Anforderungskontexte (Kenntnisse, Fertigkeit, Fähigkeit);
- Die Erschaffung und Gestaltung innovativer Konzepte und Problemlösungen;
- Die anschlussfähige Kommunikation von wissenschaftlichen Wissensbeständen, Konzepten und Methoden;
- Die Selbstregulation und Reflexion des eigenen problemlösungs- und erkenntnisgeleiteten Handelns.

Nach Meinung des Autors wurde seinerzeit insbes. im Fachhochschulbereich der Bereich der Kenntnisse, Fähigkeiten und Fertigkeiten inhaltlicher Art immer noch überbetont, es zählte „der Stoff". Hier möchte der Autor dazu beitragen, auch die weiteren Anteile des akademischen Kompetenzbegriffes angemessen mit in die Gestaltung der Curricula sowie der einzelnen Module der Informatik einzubringen.

Ein weiteres Argument gegen die Kompetenzorientierung zielt gerade vor dem Hintergrund hoher Studierendenzahlen auf die ökonomische (Durchführungs-)Ebene. Gerade in Grundlagenveranstaltungen werden Lernziele oft rein auf Inhaltsebene bzw. auf den „unteren Taxonomiestufen" (Anderson & Krathwohl 2001) formuliert. Die Pragmatik, also das „Warum bzw. Wofür" wird außer Acht gelassen, da berufsfeldnahe Beispiele oder Probleme als zu komplex erachtet werden. In Prüfungen (i.d.R. Klausuren) werden dann eher Wissens-, Verständnis- und algorithmisch lösbare Handlungsfragen gestellt, die eine fest umrissene „Musterlösung" und damit eine „objektive" Aus- und Bewertung ermöglichen.

Hier ermuntert der Autor dazu, zunächst einmal die Lernziele der eigenen Veranstaltung ehrlich, mit hoher Formulierungsqualität, nach dem Dreiklang „Was, Womit, Wozu?" und taxonomisch eindeutig zu formulieren. Wenn diese eben tatsächlich nur auf den „unteren Kompetenzstufen" liegen und das so ausreichend auf die curricularen Lernziele „einzahlt", ist es OK. Sollen aber auch Lernziele auf höherer Taxonomiestufe erreicht werden, muss oft der Stoffumfang gekürzt und auf andere Lehr- und Prüfmethoden zurückgegriffen werden.

Im Sinne des „Constructive Alignment" untersuchten wir zunächst, ob sowohl die Lehrveranstaltung als auch die Prüfung(en) konsequent auf die zu erreichenden Lernziele hin ausgerichtet sind. Insbesondere bei der Bewertung der Aufgabenlösungen

[2] S. A. die aktuelle Kritik an der Kompetenzorientierung in der gymnasialen Oberstufe und der zentralen Abiturprüfung in (Weiss & Kaenders, 2018).

und der Präsentationen konnten wir deutlichen Verbesserungsbedarf ausmachen.

Hier wurde die bisherige, eher feingranulare und quantitative Bewertung der Aufgabenlösungen zu Gunsten eines gröberen Bewertungsrasters aufgegeben (Tabelle 5). Das zu grobe Bewertungsraster für die Präsentationen inklusive der zu erstellenden Präsentationsfolien hingegen verfeinerten wir zu dem in Tabelle 6 gezeigten fünfstufigen Raster.

Mit diesen Handreichungen erfolgt seitdem die Bewertung in den zwei Schritten 1.) Beobachten und 2.) Bewerten (vgl. Szczyrba, Wildt & Dany 2008). Die Benotung wird seitdem von den Studierenden als objektiver und nachvollziehbarer angesehen.

Als weitere Änderung wurden die Präsentationstermine wieder am Ende des Semesters geblockt, so dass alle Studierenden auch für diesen Prüfungsteil die gleichen Voraussetzungen haben.

Inhaltlich haben wir insbesondere bei der Ablauforganisation die agilen Vorgehensweisen einbezogen, da diese in der Praxis zunehmend eingesetzt werden. Im Rahmen der Aufgabe zur Vorgehensmodellierung müssen die Studierenden im Kontext ihres Projektgegenstandes diskutieren, ob das von ihnen „zugeschneiderte" V-Modell XT oder ein agiles Vorgehen zielführender erscheint.

Letztendlich führten wir, um die Inhaltsvalidität der Prüfung zu erhöhen, ab dem Wintersemester 2016/17 einen einstündigen Wissenstest ein, in dem die erlangten Kenntnisse und (Technik-)Fertigkeiten der Studierenden „in der Breite" beobachten. Hier verwenden wir das MC-Format als effektives Prüfungsformat für Kenntnisse und Fertigkeiten bis zur vierten kognitiven Ebene des Wissenserwerbs („Analysieren", s. Anderson & Krathwohl et al. 2001, Miller, Linn & Gronlund 2008).

Tabelle 4 fasst die Entwicklung des Moduls PM von 2003 bis heute zusammen.

Entwicklungsphase	Auslöser	Lehr-/Lernarrangement	Prüfungsform
I.) 2003: Inhaltsorientierung und Single-Teacher-Setting	Erste PM-Veranstaltung des Autors nach der Berufung, Diplomstudiengang „Allgemeine Informatik"	Single-Teacher, Praktikum eher i.S. von Übungsaufgaben.	Klausur
II.) 2004-2006: Realitätsnähe und Team-Teaching	Gemeinsame PM-Veranstaltung mit zwei Kollegen für alle 4 Informatik-Diplom-Studiengänge der FH Köln	Team-Teaching, Praktikum mit vorgegebenem Projektgegenstand, studiengangsübergreifende Teams, Abnahme am Ende der Vorlesungszeit	Klausur
III.) 2007-2014: Lernziel-Orientierung und veranstaltungsbegleitende Prüfungsformate	Umstellung auf Bachelor und Akkreditierung, Formulierung des Lernziels	Team-Teaching, Praktikum mit selbst gewähltem Projektgegenstand, studiengangsübergreifende Teams, Meilenstein-Abgaben und Präsentationen in der Vorlesungszeit	Schriftliche Dokumentation der Lösung einer Aufgabe, Präsentation und mdl. Kurzprüfung Protokolle/Teamleistung
IV.) Seit 2015: Kompetenzorientierung	Reakkreditierung und neue Erkenntnisse zu kompetenzorientierter Lehre und Prüfung	Team-Teaching, Praktikum mit selbst gewähltem Projektgegenstand, studiengangsübergreifende Teams, Meilenstein-Abgaben und Feedback-Gespräche in der Vorlesungszeit, Präsentationen geblockt am Ende der Vorlesungszeit	Meilenstein-Einhaltung, PM-Artefakte Präsentation und mdl. Kurzprüfung, MC-Wissenstest

Tabelle 4 Entwicklungsphasen des Moduls Projektmanagement

	Team		1		4	
AT4-AY9 Bewertung! Kriterien Schlüssel in Komment. bei Aufg.Nr.	Projekt		auto_lo		Living Reality	
Aufgabe	**Bewertungskriterien**	**Faktor**	**Kommentar**	**Note**	**Kommentar**	**Note**
1	Lastenheft	0,5	G& 19F& ?Do 1So 1U+ stdN+ 10T+ 0FR!	2,4	G+ 20F+ 3D+ 0S! 3U& stdN+ 0T! 2FR&	2,5
Lastenheft	Eindruck	0,5	Deckbl. unvollst., D in use cases, sauber. Fremdprod. Fehlen!	1,7	Personae gut, Fremdp. Unstrukt. Abb-Quelle?	1,7
_Lastenheft.pdf	**Inhalt**	-	35S, zt sehr ausführlich	**2,0**	14 S., Seitennum., Links!	**2,1**
	Ampelfeedback und Note		:-)	2,0	:-)	2,1
2	FPA, CoCoMo	0,5	(460RFP 50SI 529FP 60PMfp) 11KLOC (Cbs CmdOrg Cim)+ (30PMco 9TDEVc)+	1,8	(212RFP 44SI 231FP 13PMfp)& 21KLOCo (Cbs ?Cmd ?Cim)o (60PMc 11TDEVc)o	2,5
Aufwandsschätzung	Eindruck	0,5	Fundiert, FPA u CoCoMo tw. Begründung	1,3	FPA gut, etw. umstdl., Coc basis, unvollst.; Diff nicht diskutiert	2,3
	Inhalt	-	Gut strukturiert	1,5		2,4
	Ampelfeedback und Note		:-)	1,5	:-|	2,4
4	RisikoAnalyse	0,5	(2PdR 8PjR)+ Prio Mit+ NtfP+ AE&	2,0	(#PdR #PjR) Pri Mit NtfP AE	0,0
Risikomanagement	Eindruck	0,5	Fundiert, aber keine RPZ-Berechnung. Zu wenig Produktrisiken.	2		
	Inhalt	-	knapp	2,0		0,0
	Ampelfeedback		:-)	2,0		0,0

Tabelle 5 Bewertungsschema Hausarbeiten im WiSe 2017/2018 (Ausschnitt)

PROJEKTMANAGEMENT WS 2017-2018 Bewertungskriterien für Folien und Präsentation			
MW 20171114			
	Bereich	**Note**	**Kriterien**
F Inhalt	**mangelhaft**	5	Inhaltlicher Zusammenhang und Aufgabenbezug nicht erkennbar
	Ausreichend	4	Inhaltlicher Zusammenhang und Aufgabenbezug erkennbar, aber implizit
	Durch-schnittlich	3	Inhaltlicher Zusammenhang, Aufgabenbezug und roter Faden erkennbar, Ergebnisse werden vermittelt
	Über dem Durchschnitt	2	Kurze und im Wesentlichen nachvollziehbare Darstellung, Aufgabenstellung und Ergebnisse werden klar vermittelt
	Exzellent	1	Kreative, jederzeit nachvollziehbare Darstellung von Zielen, Aufgabenstellung und Ergebnissen
F Darstellung	**mangelhaft**	5	Themenstellung verfehlt, viele nebensächliche Details
	Ausreichend	4	Einige Inhalte erkennbar, ausschweifend oder viel zu kurz
	Durch-schnittlich	3	Erläuterung einiger Ergebnisse und Details
	Über dem Durchschnitt	2	Kurze Erläuterung einiger wichtiger, begründet ausgewählter Ergebnisse, Darstellungsformen korrekt, Details sinnvoll
	Exzellent	1	Prägnante Erläuterung, Begründung und eigene Interpretation der zentralen Ergebnisse, Details helfen bei Verständnis

Tabelle 6 Bewertungsschema Präsentation im WiSe 2017/2018 (Ausschnitt)

Abb. 6 Evaluierungsergebnisse aus dem Wintersemester 2016/17 (Ausschnitt)

Wir führen das Modul PM nun seit dem Wintersemester 2016/17 in der oben beschriebenen kompetenzorientierten Form durch. Einige Ergebnisse der ersten studentischen Evaluierung nach der Umstellung sind in Abb. 6 zusammengefasst.

Ausblick

Nach wie vor ist es für die Studierenden schwierig, sich für das Modul PM in neuen, studiengangsübergreifenden Teams zusammen zu finden und möglichst zügig als Team zu arbeiten.

Viele Ergebnisse zeigen, dass für das Projektmanagement die Simulation von Projekten einen guten Einstiegspunkt darstellt (Mandl-Striegnitz, 2001 und die vielen Folgeveröffentlichungen zu SESAM). Erste Versuche mit SESAM waren zwar ermutigend, wurden aber eher im „single-player" Modus in der Rolle „Projektleitung" absolviert und dienten dabei weniger der Sensibilisierung der Studierenden für die Arbeit in einem Team.

Speziell für den Bereich der Teamarbeit konzipieren wir zur Zeit mit einer Kollegin und einem Kollegen aus dem Betriebswirtschaftlichen Institut Gummersbach ein Simulationsspiel zur Einführung in die Dynamik der Teamarbeit (Stumpf & Thomas 2003). Hiermit sollen die Studierenden einerseits auf die bevorstehende Teamarbeit bei der Bearbeitung der PM-Projektaufgaben vorbereitet werden. Andererseits möchten wir sie damit für kritische Teamprozesse in der Praxis sensibilisieren.

Danksagung

Der Autor dankt den Kollegen Holger Günther und Lutz Köhler sowie unseren Mitarbeiterinnen und Mitarbeitern Beate Breiderhoff, Konstantin Dimitriou, Guido Münster, Alex Maier, Patrick Odenwald, Beate Otztronsek, Uwe Poborski, Pascal Schönthier und Marc Schwede für die vielen lebendig geführten Diskussionen und das konstruktive Miteinander bei der Durchführung und Entwicklung unserer Veranstaltung *Projektmanagement*! Wertvolle Unterstützung bei der Umsetzung der kompetenzorientierten Lehre und Prüfungformen (Constructive Alignment) in Phase IV leistete das Zentrum für Lehrentwicklung der TH Köln, insbesondere Susanne Gotzen, Birgit Sczyrba sowie Oliver Reis von der Universität Paderborn während der Multiplikatoren-Ausbildung zu kompetenzorientierten Prüfungen.

Literatur

Aichele, C. &. S. M. (2014). IT-Projektmanagement. Berlin: Springer Vieweg.

Anderson, L. W.; Krathwohl, D. R. et al. (2001). A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. Complete Edition. New York, Pearson/Longman.

Balzert, H. (1996). Lehrbuch der Software-Technik (Bd. 1). Software-Entwicklung. 1. Aufl., Spektrum-Akademischer Verlag, Heidelberg.

Balzert, H. (1998). Lehrbuch der Software-Technik (Bd. 2). Software-Management. 1. Aufl., Spektrum-Akademischer Verlag, Heidelberg.

Biggs, J.; Tang, C. (2011). Teaching for Quality Learning at University. Maidenhead, Open University Press/McGraw Hill.

Böttcher, A. & Thurner, V. (2011). Kompetenzorientierte Lehre im Software Engineering. Proc. SEUH 10. dpunkt.verlag, Heidelberg.

Broy, M. & Kuhrmann, M. (2013). Projektorganisation und Management im Software Engineering. Berlin: Springer Vieweg.

Feyhl, A. & Feyhl, E. (1996). Management und Controlling von Softwareprojekten. Gabler Wirtschaftsverlag, Wiesbaden.

Fleischmann, A. & Spies, K. (2005). Teamtraining für Software-Ingenieure. Proc. SEUH 9. dpunkt.verlag, Heidelberg.

Friedlein, A. (2003). Web-Projektmanagement. dpunkt.verlag, Heidelberg.

Grupp, B. (1998). Qualifizierung zum Projektleiter: DV-Projektmanagement im Wandel. 4. Auflage, Computerwoche-Verlag, München.

Henrich, A. (2001). Management von Softwareprojekten. Kurs 1895. Hagen: FernUniversität.

Hummel, O. (2013). Transparente Bewertung von Softwaretechnik-Projekten in der Hochschullehre. Proc. SEUH 11. dpunkt.verlag, Heidelberg.

Kerzner, H. (2009). Projekt-Management. Bonn: mitp-Verlag.

Mandl-Striegnitz, P. (2001) Qualifizierte Software-Projektmanager durch simulationsbasierte Ausbildung. Proc. SEUH 7. dpunkt.verlag, Heidelberg.

Miller, M. D.; Linn, R. L. & Gronlund, N. E. (2008). Measurement and Assessment in Teaching. (10th Edition), New York, Pearson Education Ltd.

Reis, O. (2010). Kompetenzorientierte Prüfungen – Wer sind sie und wenn ja wie viele? In: Terbuyken, G. (Hg.) In Modulen lehren, lernen und prüfen.. Loccum: S. 157-183.

Stumpf, S. & Thomas, A. (Hrsg.) (2003) Teamarbeit und Teamentwicklung. Reihe: Psychologie für das Personalmanagement - Band 22, Hogrefe, Göttingen.

Szczyrba, B.; Wildt, J. & Dany, S. (2008). Prüfungen auf die Agenda! Bielefeld, AHD/wbv, Verlag W. Bertelsmann.

Thurner, V.; Böttcher, A.; et al. (2015): Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen. Proc. SEUH 12. http://ceur-ws.org/Vol-1332/

Walzik, S. (2012). Kompetenzorientiert prüfen: Leistungsbewertung an der Hochschule in Theorie und Praxis, Opladen & Toronto, UTB GmbH.

Weiss, Y.; Kaenders, R. (2018). Die Kompetenzfalle. Spektrum der Wissenschaft, Ausgabe 9/18, S. 80-85.

Future Skills: How to strengthen computational thinking in all software project roles

Gudrun Socher*, Sarah Ottinger*, Veronika Thurner*, Ralph Berchtenbreiter[o]
*Department of Computer Science and Mathematics | [o]Department of Tourism
Munich University of Applied Sciences, <firstname>.<lastname>@hm.edu

Abstract

The digital transformation leads to software systems pervading almost all spheres of private and professional life. To ensure that these software systems are designed and successfully implemented as needed, intensive collaboration is essential in the key roles in software projects, in particular for the roles of product owner, user experience designer, as well as software engineer. The collaboration of people with usually different levels of IT-savviness requires the appropriate skills of those involved, which are also called Future Skills. Computational thinking is an important skill for everyone involved in software projects, no matter which role they are in.

We describe an interdisciplinary tool-based teaching-and-learning program where we build virtual voice-based assistants (voice apps for Amazon Alexa) in interdisciplinary student teams to train computational thinking and collaboration skills. A first competency test validates the effectiveness of our approach.

Motivation: Future Skills[1]

In current digitalization initiatives, there is a lot of discussion on how to increase graduation numbers in software engineering related study programs in order to have more skilled people driving the ongoing digital transformation. In this discussion, however, we often forget that digitalization is always related to an application domain. The digital transformation benefits strongly if software-related skills are strengthened not only for the core software engineering roles, but also for less-technical roles in software projects like product owners, or user experience designers (see Figure 1), as well as for even more general roles such as product management or project management.

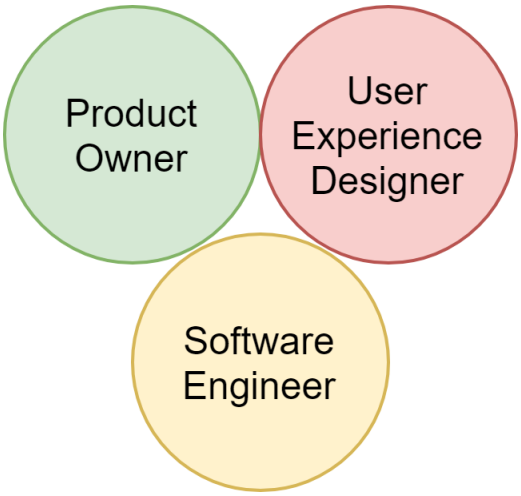


Figure 1: Key roles in software projects.

Software products can only be successful if the key roles work hand in hand in software projects: product owners with their knowledge of the application domain and product vision, user experience designers who guide human-computer interaction, and software engineers being responsible for software implementation. The skills and competencies related to these roles are essential in successful projects. They are required to successfully meet the challenges of digital transformation.

How do we best train the talents for the ongoing digital transformation, and what exactly do future employees need to learn? Stifterverband, a joint initiative of German organizations, has published a discussion paper in September 2018 together with McKinsey & Company where they address three core competency

[1] With 'Future Skills' we refer to 'competencies' required by university graduates across all majors in the coming years. These competencies are necessary to meet digital requirements as they are currently expected in business and society (cf. (Kirchherr et al., 2018)).

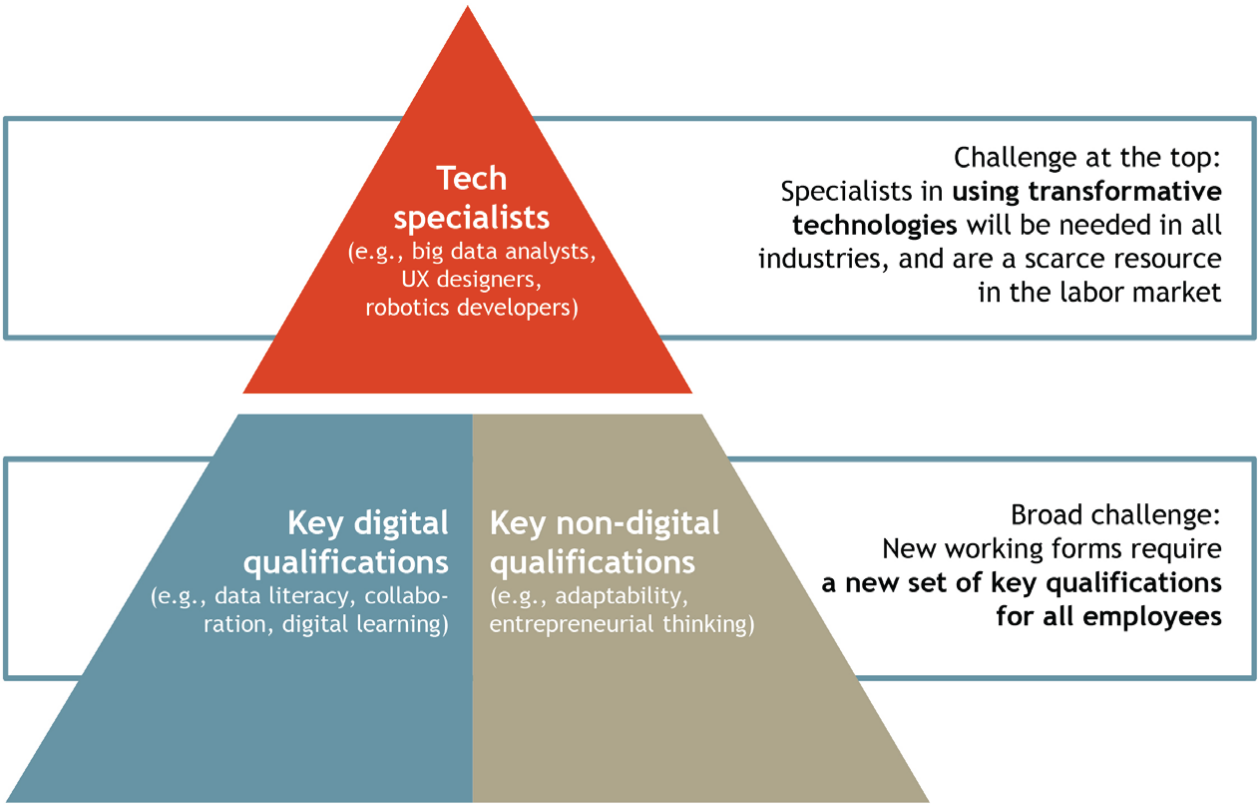


Figure 2: Future Skills: Digitalization and new job profiles lead to two challenges for companies. (1) Jobs shift towards IT related profiles. (2) Work processes and work requirements change for the majority of employees. Many employees thus need a modified set of digital and non-digital key competencies (cf. (Kirchherr et al., 2018)).

categories driving the digitalization (Kirchherr et al., 2018). Figure 2 illustrates competencies related to the digital transformation as well as to new forms of work.

With changes in the job portfolios and new forms of work, there has been an expected shortage of qualified people for some time now. In particular, the job market will be increasingly dominated by job profiles that are heavily related to software engineering. Accordingly, a lot of effort is spent on increasing the number of graduates of software engineering related degree programs, in order to increase the number of software engineers in the market and thus to meet the growing demand for qualified specialists.

At the same time, it is essential to strengthen everybody's digital and non-digital key competencies. Digitalization is not just an IT issue. Digitalization is inherently linked to the digital transformation and thus the creation of digital automation or digital assistance systems in an application domain. Therefore, it is just as important for employees and students of all application domains to acquire software engineering competencies.

Stifterverband (Kirchherr et al., 2018) stresses that both digital and non-digital key competencies are a must-have for all current students. Key digital skills include - among others - data literacy, collaborative skills and digital learning skills. Based on Wing (2006), computational thinking can be considered as a prerequisite for digital learning. By now, the relevance of all these skills for virtually any member of our professional work force is undisputed and widely recognized.

How can we integrate the development of these competencies into study programs? Curricula are often overfull and the amount of available (and often essential) knowledge is increasing in almost every application domain. Solutions have to be found to systematically integrate key competencies required for the digital transformation into the teaching-and-learning processes without weakening the core foundation in the respective application domains.

Project-based learning is a successful instructional learning format that has been proven to systematically strengthen some of the required future skills (Bell, 2010). Interdisciplinary project-based learning is even more effective, as team diversity is an additional success factor for creative, goal-oriented collaboration in addition to the future skills mentioned above (Digitalisierung, 2016; Meier et al., 2007).

Related work

Wing (2006) introduced the importance of computational thinking for computer science-related tasks 12 years ago. She defines computational thinking as the interplay of decomposition and abstraction and recommends strengthening computational thinking in all study programs.

The ability of abstract thinking, in turn, has long been recognized as a key competency of many technical disciplines, especially for computer science (Bucci et al., 2001; Kramer, 2007) and software engineering (Ghezzi et al., 2002). A distinction is made between static and dynamic abstraction, i.e. the abstraction of structural entities (static) and of processes or behavior (dynamic) (Davis et al., 2014).

The central element of computational thinking is a problem-oriented (as opposed to a solution-oriented) approach (Lorenz and Wurzer, 2014). It is essential to get to the root of a problem or task, to abstract it and to understand contexts and regularities. The goal is to reduce the complexity of the task (keyword: decomposition (Wing, 2006)) and to systematically limit the choice of possible solutions. Only then are potential solution components identified and abstracted into an overall behavior. Computational thinking requires not only the ability to decompose, but also the ability to abstract behavior (Wing, 2006), thus requiring a very high degree of dynamic abstraction in particular. Since algorithms always work on data entities, a corresponding degree of static abstraction is necessary.

In teaching-and-learning practice, it can be observed that not all students have a sufficient level of abstraction and computational thinking to be able to cope with the study program requirements. This is especially true for students of subjects related to computer science. Accordingly, various approaches have been developed to systematically strengthen these abilities (Hazzan and Kramer, 2007; Böttcher et al., 2016). These approaches mainly focus on promoting computational thinking in students of computer science related majors, but do not provide teaching-and-learning concepts for strengthening these skills in students of non-technical subjects, where little to no IT-affinity can be expected.

Goals

In this work, we develop and apply a teaching-and-learning program for promoting computational thinking in students – and, as an essential basis for this, for fostering students' static and dynamic abstraction competency. To this end, we develop a teaching-and-learning program for interdisciplinary, project-based learning that addresses computer science students as well as non-IT students. Our concept takes into account our students' prior knowledge as well as their individual learning requirements.

In our teaching-and-learning program, we create an easy introduction to digital projects for interdisciplinary student teams. To achieve this, we use a tool-chain for creating voice-based virtual assistants (such as Amazon Alexa), as well as by using Github. Over the last years, the usability of many tools for creating software systems has evolved and improved significantly. Digital tools in the context of software engineering are nowadays no longer editors that are specifically tailored to computer nerds. Rather, nowadays they often are web-based interactive tools that are fun to use and that guarantee good results even for people without IT- and software engineering skills.

Our student teams include tourism majors and computer science students. Github is used as a source code repository by computer science students, as well as a ticketing and project communication tool for all students. We furthermore use the Github project board as a virtual agile board. In this way, we enable all students to make an active, creative contribution in a digital software project even without programming knowledge.

The non-IT students (i.e. students of an application domain) are either in the role of *product owner* or *user experience designer*. The role of *product owner* includes gathering and aggregating the users' needs and desires which requires static abstraction capabilities. The *user experience designer* role focuses on reflecting what users expect. This definitely requires consideration of dynamic processes. In parallel, computer science students deepen their experience in the role of *software engineer* by using new technologies for the implementation of voice-based assistants.

We use pseudonymized pre- and post-tests to analyze to what extent our interdisciplinary tool-based teaching-and-learning program actually fosters the addressed competencies of static and dynamic abstraction in the students.

Computational thinking and voice-based assistants

Voice-based virtual assistants (voice apps) are currently being massively pushed by major software companies. Examples are Amazon Alexa, Google Assistant, Cortana, and many more. In particular, Amazon and Google provide well-designed, web-based tools that developers can use to create new voice apps with their cloud offerings. These tools and cloud offerings are available free of charge for educational use. In addition, the tools are so sophisticated that it's fun to play with them. In just a few minutes, cool voice-user-interfaces can be created without previous knowledge, so first success can be achieved quickly. Figure 3 shows the Alexa Developer Console, a tool for creating voice apps for Amazon Alexa.

To develop an application for voice-based assistants, the development team must design the Voice User Interface (Voice UI) and implement the business logic. A voice UI must be structured in such a way that the dialogue appears natural to the users. At the

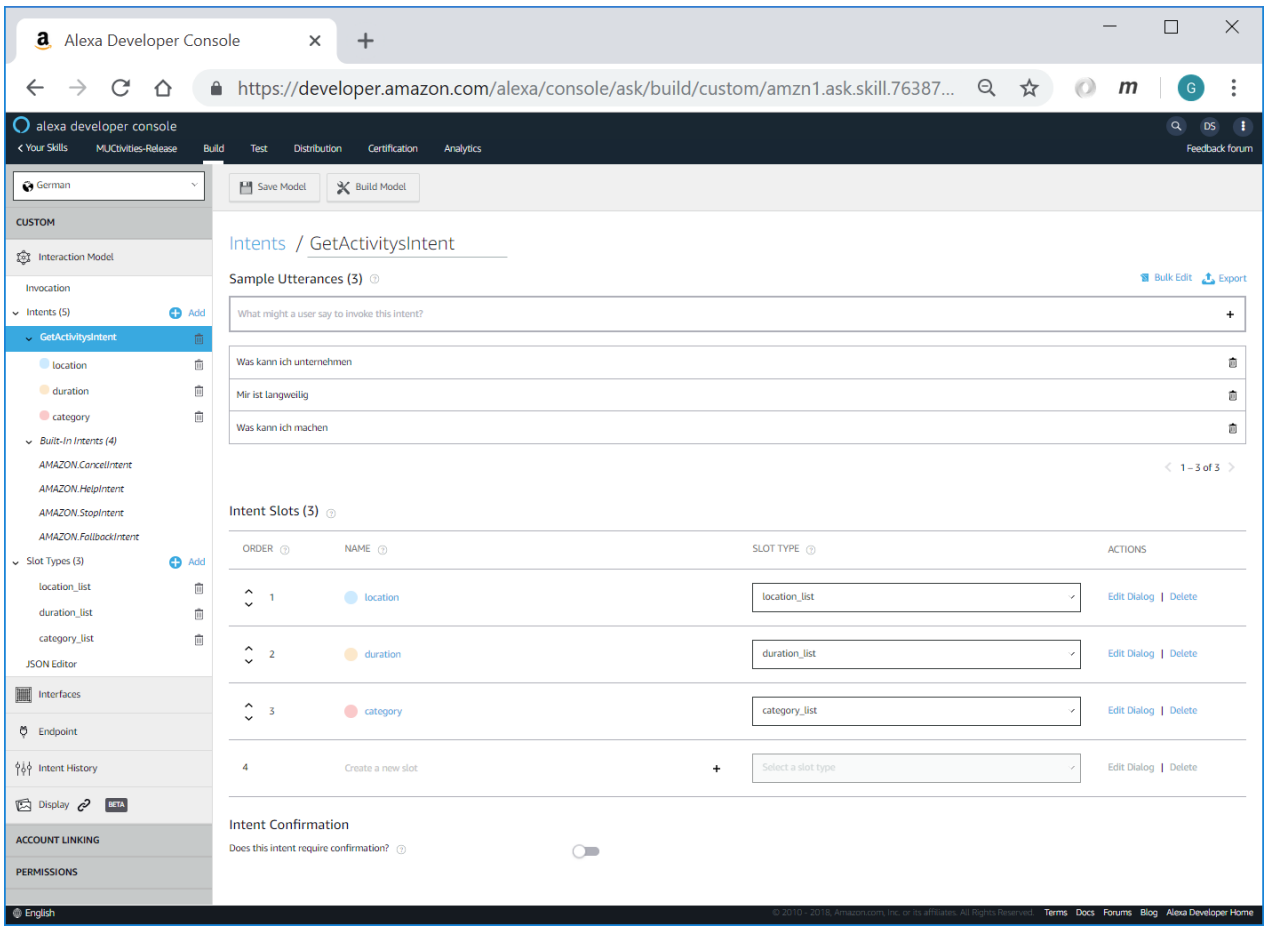


Figure 3: Alexa Developer Console: Web-based tool to create voice apps for Amazon Alexa.

same time, the dialogue must be designed so that the goals expressed by the users or the intentions behind them can be clearly identified and assigned to the implemented business logic. In the jargon of voice UIs, these intentions are called *intents*. Each intent must clearly invoke a feature of the implemented business logic.

Figure 4 shows an example dialog with Alexa for an application called the "joke-of-the-day". A person starts the voice app and gets told one joke. More jokes can be requested. If no more joke is desired, the voice app closes.

Structuring a dialogue into intents strengthens both static and dynamic abstraction abilities.

The identification of the individual intents in a dialogue requires a static abstraction. For example, the joke-of-the-day application is structured into the following intents and the resulting dialog steps:

- welcome
- joke
- closing

The individual dialog steps or intents must then be arranged in a meaningful sequence (the dynamic behavior). This process is rather simple for the joke-of-the-day application (see Figure 5).

The design of a dialog for a voice-based virtual assistant is thus well suited to train both static and dynamic abstraction skills, and to guide students towards computational thinking. Therefore, we use dialog design and implementation for a voice-based assistant as a task for an interdisciplinary tool-based teaching-and-learning program to strengthen computational thinking.

Interdisciplinary project to strengthen computational thinking

Our didactic concept for the promotion of computational thinking and communicative future skills structures the learning process into several phases (see Table 1). Initially, the students are taught core concepts in non-interdisciplinary groups.

More precisely, the computer science students first learn the technical basics of voice-based assistants, and are introduced to tools for designing and implementing them. Furthermore, computer science students learn requirements engineering. Having that down their belts, these students are well equipped to

Task	**Computer Science Students**	**Students of Application Domain**
Introduction to voice-based assistants	Examples and tutorial for creating a first voice app	Examples and simulation of dialogues between two partners
Idea for voice app and its structure	Invocation, intents, slots	Generating ideas and defining MVP
Concept for Voice App	Requirements engineering	Specification of a first dialogue
Using tools	Alexa Developer Console, Github	Invocable, Alexa Developer Console, Github
1st Pitch: Students of application domain pitch their ideas to find computer science students for their developer team.		
1st Sprint	1st Release	Refining the dialogues
2nd Sprint	2nd Release	Test 1st Release → Change Requests
2nd Pitch: Voice app demos by computer science students.		
3rd Sprint	3rd Release	Test 2nd Release → Final changes and improvements
3rd Pitch: Final presentation with guests.		

Table 1: Structure of the teaching-and-learning process throughout the semester for our interdisciplinary tool-based teaching-and-learning program for developing Alexa voice apps.

take on the role of *software engineer* in the interdisciplinary teams that are formed later on.

The task of the application domain students is to develop an idea for a voice-based assistant (in this case an Alexa voice app). For this idea, they then define the dialogue between the user and the voice-based assistant required for a Minimum Viable Product (MVP). This dialog thus contains at least those steps and procedures that are necessary for the minimal functional implementation of the idea. It is important to break down the dialogue into short, simple steps and to structure it. The complexity of creating the voice app is significantly greater than the example of the "joke-of-the-day" shown in Figure 4. Suitable ideas for voice apps are (quiz) games, guides, or useful assistants.

The dialogue is tested and tuned by the students of the application domain. Then, using Invocable[2], the students of the application domain create a first interactive but hard-coded prototype of the voice app.

The interdisciplinary collaboration in the teams begins when the students of the application domain present their ideas to the computer science students. The students of the application domain pitch the prototypes of their voice-based assistants to computer science students and try to motivate them to join their development team. Following these pitches, mixed teams are formed each consisting of computer science students and students of the application domain.

Within the interdisciplinary teams, the computer science students give feedback to the students of the application domain on the design of the dialog that underlies the respective prototype. Based on this, the voice user interface is subsequently refined together. The computer science students take their "natural role" as *software engineer* in the interdisciplinary project. The application domain students, on the other hand, are both *product owner* and *user experience designer*. So they design the interaction between the user and the voice-based assistant in such a way that this interaction is technically meaningful and needs-based from their own perspective. So far in our didactic concept, user experience design is not explicitly taught due to lack of time. However, it would be desirable to improve this in the future.

Computer science students implement the back-end, while students of the application domain are responsible for the front-end in the implementation phase of the voice-based assistant. A simple form of Scrum is useful for organizing the development process into sprints, thus structuring the semester process. Github repositories, including the integrated project boards and the integrated ticketing system (Github issues),

[2]Invocable: `https://www.invocable.com`

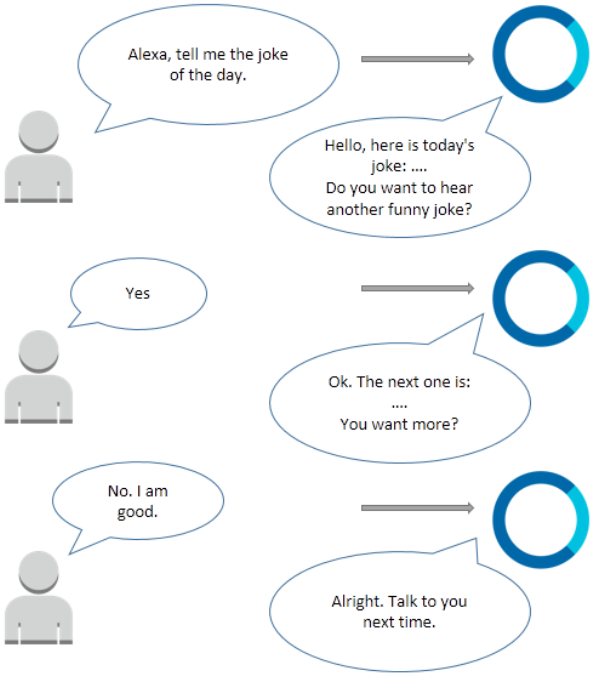


Figure 4: An example dialogue with a voice-based virtual assistant telling jokes.

support team collaboration through an appropriate tooling infrastructure.

This teaching-and-learning program was used for the first time in the winter semester 2018/19 at Munich University of Applied Sciences. Four computer science students (3rd semester) and two to three students of tourism management (6th semester) form a mixed team for the pilot run of our program.

Even if static and dynamic abstraction are not explicitly addressed, all students in this one-semester interdisciplinary tool-based program train their static and dynamic abstraction abilities by structuring and specifying the dialogue between a person and the voice-based assistant in such a way that a corresponding Alexa voice app can be built. All students (including application domain students) work with the Alexa Developer Console and Invocable tools. The use of tools enables all students to work with a working interactive voice app. The working prototype provides rapid feedback so that in particular the students of the application domain can immediately check their dialogue structuring.

From our perspective, this interdisciplinary tool-based teaching-and-learning setting is well suited to foster our students' abstraction skill and more effective in this area than regular class exercises. Furthermore, standardized processes, such as completing Github Issues in team communication, help to structure collaboration.

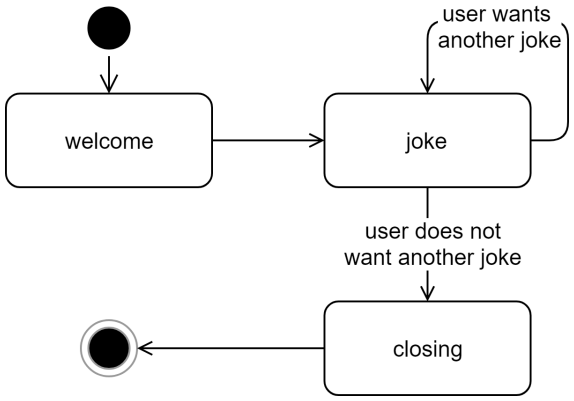


Figure 5: Dynamic structure of the example application joke-of-the-day.

Assessing computational thinking skills

As we were interested in investigating the development of our students' computational thinking skills during the interdisciplinary tool-based teaching- and learning program, students were requested to work on a competency test at the beginning (week 2) and at the end of the semester (week 11). The test covered two facets of computational thinking, namely static and dynamic abstract thinking processes.

The test was taken by two groups of students, all of which were working on voice-based virtual assistants in a project based way. The first group are computer science students and students of the application domain who worked together in interdisciplinary teams. The other group, our control group, consisted purely of computer science students.

All these students were asked to solve two tasks that both form our competency test and that require static and dynamic abstract thinking processes, respectively. Using pseudonyms allowed us to match the students' pre- and post-tests and thus, to analyze their individual learning outcomes. (Regarding the tourism-management students' test performances, we will discuss the results in the next section "Coding scheme of the abstract thinking test & first results". The test performances of the computer science students have also been analyzed, but will not be presented in this paper.)

Our hypothesis is that the computational thinking skills of students that actively participated in our interdisciplinary tool-based program increased significantly. More specifically, we assume that all students benefit from developing and implementing their innovative ideas on Alexa voice apps, as both the design and the creation of voice-user-interfaces require static and dynamic abstract thinking processes. Note that even though we expect that students working in interdisci-

plinary teams strongly train their collaboration skills during the course, we have not explicitly assessed theses skills in a test.

The first task of our competency test includes a menu of 22 different coffee specialties and requests the students to teach a new barista about the coffee recipes. The menu contains pictures and ingredient lists for each coffee specialty. The challenge is to structure the various coffees and their ingredients in such a way that a new barista can quickly grasp and learn them. In the second task, students are asked to generalize the abstraction process they applied when solving the first task.

To successfully accomplish the second task, the participants first have to become aware of their own actions, identify and structure those processes, and derive a procedure that would be transferable to similar tasks. Therefore, they have to dynamically abstract from their own approaches and accurately document their solutions. From our perspective, task 1 mainly deals with static abstract thinking processes, whereas task 2 covers dynamic abstract thinking processes.

Students are allowed 20 minutes for working on the first task and 15 minutes for completing the second task. Once the students begin working on task two, they are not allowed to use the documents they have generated in task one. In winter semester 2018/2019, 18 tourism students and 54 computer science students participated in the computational thinking competency test.

Coding scheme of the computational thinking competency test & first results

To analyze the competency test of computational thinking skills, a comprehensive coding scheme was developed incorporating five criteria to evaluate static abstract thinking processes (S1-S5) and three criteria to measure dynamic abstract thinking processes (D1-D3). We defined two additional criteria operationalizing one's ability for self-reflection (R1) and for identifying the requirements needed (R2). All criteria differentiate between four levels of competency (outstanding, good, satisfactory, not yet satisfactory). Score 1 characterizes the lowest level of competency, score 4 the highest one.

We attempted to capture static abstract thinking processes by evaluating students' performances along these four criteria:

- Criterion S1: developing categories that ideally simplify the given representations (of coffee specialties) and structuring ingredients;
  Levels of competency may be described as 'unrefined', 'put together', 'structured' and 'organized' (Hershkowitz et al., 2001).
- Criterion S2: identifying and parameterizing attributes.
- Criterion S3: presenting categories in a structurally-sound way, e. g. as UML-diagrams.
- Criterion S4: using formal notations in a logically consistent way.
- Criterion S5: recognizing familiar structures and realizing that the structures are coherent in a given situation.

Dynamic abstract thinking processes are operationalized by defining these three criteria:

- Criterion D1: identifying a coherent chain of processes.
- Criterion D2: presenting a coherent chain of processes in a structurally sound way.
- Criterion D3: relating the identified chain of processes to one's own solution, e. g. using one's own approach as a basis for generalizing and inferring appropriate processes.

We considered two additional criteria, characterizing one's ability for self-reflection as well as metacognitive thinking skills:

- Criterion R1: identifying errors and reflecting one's own approach.
- Criterion R2: taking the users' preferences into account, as well as the requirements.

First results indicate that the computer science students demonstrate a significantly higher initial level of static abstract thinking skills (mean-value $M=2.04$; standard-derivation $SD=0.49$), in comparison to the tourism majors ($M=1.77$; $SD=0.46$) ($t=2.119$; $df=70$; $p=0.030$).

Regarding the students' initial dynamic abstract thinking skills, we found no statistically significant differences between the performances of computer science students ($M=2.12$; $SD=0.51$) and tourism students ($M=2.28$; $SD=0.43$; whereby $U=384.00$; $p=0.210$). It seems that tourism students have reached a slightly higher level of dynamic abstract thinking skills than computer science students.

Furthermore, the results indicate that computer science students ($M = 2.07$, $SD = 0.78$) and tourism students ($M = 1.81$, $SD = 0.73$) do not differ significantly in their self-reflection skills ($U = 384.00$, $p = 0.210$). We observe only a slight tendency in favor of the computer science students.

Overall, the competency test gives some insight that most of the students – computer science students (3rd semester) as well as tourism students (6th semester) – struggle with demonstrating their static and dynamic abstract thinking skills. According to Wing (2006) and based on our results, we strongly recommend the integration of interdisciplinary teaching- and learning-programs in students' curricula to foster students' computational thinking skills.

Regarding students' reflection skills, it seems to be important to encourage students to reflect their own approaches and to think about the requirements needed.

The results of the pre- and post-tests provide empirical evidence that the tourism students' static abstract thinking skills development can be characterized by a significant increase ($t = -3.986$, $p = 0.001$) supporting our hypothesis. After actively participating in the interdisciplinary tool-based teaching-and-learning program ($M = 2.21$, $SD = 0.48$) the tourism students score significantly better in terms of their static abstract thinking skills than before their participation in this program ($M = 1.77$, $SD = 0.46$). The effect size by Cohen (1992) is around $r = 0.695$, thus indicating a strong effect. Regarding the tourism students' dynamic abstract thinking skills, no significant effect can be reported ($t = -1{,}475$, $p = 0.159$). However, there is also a tendency towards an increase of dynamic abstract thinking skills ($M = 2.24$, $SD = 0.44$). Our hypothesis that tourism students improve their computational thinking skills during the interdisciplinary tool-based program can be confirmed.

Challenges and experiences

Project-based teaching presents many challenges to teachers (Barron et al., 1998), among others:

- Formulate clear definitions of learning goals and competencies students should acquire.
- Make sure that the project tasks cover the planned learning content to the desired extent and adequately demand and strengthen the competencies to be acquired.
- Build social interaction structures within the project teams that allow a balanced distribution of roles and tasks.
- Create a good (i.e. applicable) schedule.

128 students in computer science and tourism management were in the pilot group in the winter semester 2018/19. We combined a software engineering module and a module for digital marketing and management. Both modules have their own learning objectives and content. In addition to the learning objectives of these respective modules, additional learning objectives of the interdisciplinary tool-based teaching-and-learning program include the increase in static and dynamic abstraction abilities mentioned in this paper as well as the improvement of collaboration and communication skills in digital projects. Therefore, instructors need to encourage students in their learning process. At the same time, instructors have to take care to not overburden their students.

We solve the challenges of the different learning objectives and content of the initial modules by running some part of the interdisciplinary collaboration between the students in a purely virtual way, using the cloud-based Alexa Developer Console as well as Github and Github issues. The classroom events where both computer science students and students of the application domain are together are the three pitches which are highlighted in gray in Table 1, i.e. the pitch of the prototypes by the students of the application domain, the pitch of the voice app demos by the computer science students, and finally the joint pitch during the final presentation. A mix of virtual and physical collaboration creates enough space for both teaching and learning sessions to accommodate the specific contents of the respective modules and the corresponding competencies according to the definition of learning goals.

The interdisciplinary project was a lot of fun for everyone involved. For that reason alone, it is highly recommended to repeat the project. The use of new web-based development tools was well received by all participating students independent of their field of study.

The computer science students were motivated primarily by the fact that new voice technologies were used in the context of this project. In turn, tourism students have grown into the role of *product owner* during the project. Furthermore, by using the development tools, they were in able to increase their competency in using web-based tools. They also liked to creatively integrate sound effects into the Alexa voice apps.

Summary and outlook

It is important for all students to develop and to strengthen their ability of computational thinking in order to meet the requirements of the digital transformation. As a basis, it is helpful that the students first develop the skills for static and dynamic abstraction as these are a basic building block of the ability of computational thinking. Computational thinking is one of the Future Skills (Kirchherr et al., 2018), which are important core competencies for the future working life as well as for the participation in business and society in the era of the digital transformation and the new forms of work linked to it.

We strengthen computational thinking through interdisciplinary, tool- and project-based learning. As a project topic and work context, we select the design and implementation of voice-based digital assistants (so-called voice apps). The students are encouraged to use static and dynamic abstraction for the specification of the dialogue between a human and a voice-based assistant.

A competency test was developed in order to measure the effectiveness of our approach. The test was run at the beginning and at the end of the semester. (The evaluation of the post-test is not yet completed.) The pseudonymized test results are used to determine the extent to which the students were able to improve

their abilities of static and dynamic abstraction during the program.

More tests are required for a more detailed analysis of future skills through interdisciplinary tool- and project-based learning in digital projects. Accordingly, we plan to improve and further expand our approach so that additional competencies are targeted and the effects are captured by additional measuring instruments.

References

Barron, B. J., D. L. Schwartz, N. J. Vye, A. Moore, A. Petrosino, L. Zech, and J. D. Bransford
1998. Doing with understanding: Lessons from research on problem-and project-based learning. *Journal of the Learning Sciences*, 7(3-4):271–311.

Bell, S.
2010. Project-based learning for the 21st century: Skills for the future. *The Clearing House*, 83(2):39–43.

Bucci, P., T. Long, and B. Weide
2001. Do we really teach abstraction? In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, New York, USA. ACM.

Böttcher, A., K. Schlierkamp, V. Thurner, and D. Zehetmeier
2016. Teaching abstraction. In *2nd International Conference on Higher Education Advances, HEAd'16*, P. 357–364, València. Universitat Politècnica de València.

Cohen, J.
1992. Statistical power analysis. *SAGE Journals*, 1(3):98–101.

Davis, D., T. Yuen, and M. Berland
2014. Multiple case study of nerd identity in a cs1 class. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCS'14*, P. 325–330, New York, USA. ACM.

Digitalisierung, H.
2016. The digital turn – hochschulbildung im digitalen zeitalter. *Edition Stifterverband, Berlin*.

Ghezzi, C., M. Jazayeri, and D. Mandrioli
2002. *Fundamentals of Software Engineering*. Prentice Hall PTR.

Hazzan, O. and J. Kramer
2007. Abstraction in computer science & software engineering: a pedagogical perspective. *Frontier Journal*, 4(1):6–14.

Hershkowitz, R., B. B. Schwarz, and T. Dreyfus
2001. Abstraction in context: Epistemic actions. *Journal for Research in Mathematics Education*, Pp. 195–222.

Kirchherr, J., J. Klier, C. Lehmann-Brauns, and M. Winde
2018. Future Skills: Welche Kompetenzen in Deutschland fehlen.

Kramer, J.
2007. Is abstraction the key to computing. *Communications of the ACM*, 50.

Lorenz, W. E. and G. Wurzer
2014. Algorithmisches denken. In *Brickster Style – Digitales Entwerfen eines Kulturzentrums in Wien-Meidling*, P. 7–10. Technische Universität Wien.

Meier, A., H. Spada, and N. Rummel
2007. A rating scheme for assessing the quality of computer-supported collaboration processes. *International Journal of Computer-Supported Collaborative Learning*, 2(1):63–86.

Wing, J. M.
2006. Computational thinking. *Commun. ACM*, 49(3):33–35.

Informatik ist nicht nur Programmieren – aber ohne Programmieren ist nichts Informatik

Oliver Radfelder, Karin Vosseberg, Ulrike Erb, Henrik Lipskoch
{oradfelder,kvosseberg,uerb,hlipskoch}@hs-bremerhaven.de
Hochschule Bremerhaven

Zusammenfassung

Ziel der Studieneingangsphase in den Bachelorstudiengängen Informatik und Wirtschaftsinformatik der Hochschule Bremerhaven ist, die Studierenden in eine Fachkultur einzuführen, in der Informatik nicht nur Programmieren ist - aber ohne Programmieren nichts Informatik ist. In kleinen Schritten werden die Studierenden an ihr grundlegendes Handwerkszeug und eine einfache Linux-basierte Infrastruktur für die Automatisierung von Prozessen herangeführt sowie das Arbeiten in Teams in einem ersten Projekt erprobt. Mit regelmäßigen kleinen Übungsaufgaben werden die verschiedenen Grundlagenfächer miteinander verzahnt. Über den Projektkontext wird ein Anwendungsbezug hergestellt. In dem vorliegenden Beitrag wird das Konzept der überarbeiteten Studieneingangsphase vorgestellt und anhand von kleinen Beispielen die Verzahnung von Modulen demonstriert. Im weiteren wird ein Beispiel für eine Lerneinheit aus den Workshops der Studieneingangsphase beschrieben.

Abstract

The introduction phase of the bachelor programmes Informatics and Business Informatics at Bremerhaven University of Applied Sciences aims at familiarising students with a faculty culture which understands informatics not only as programming, but follows also the idea that without programming nothing is informatics. In small steps, students learn to handle their basic tools. They are introduced to a simple Linux-based infrastructure for the automation of processes, and experience collaborating in teams while working on a first project. Through continuous small exercises they get an understanding of basic topics of the first semester modules and finally apply their knowledge in the project context. This article presents the concept of the introduction phase, describes the interlocking of modules based on small examples, and shows an example of a workshop unit of the introduction phase.

Einführung

Die zunehmende Heterogenität der Studierenden im ersten Semester erfordert ein Umdenken in der Gestaltung der Studieneingangsphase. Nicht zuletzt gefördert durch den Qualitätspakt Lehre wurden in den letzen Jahren an vielen Hochschulen Projekte initiiert, die Voraussetzungen an die Fachkompetenzen, die Studierfähigkeit aber auch die soziale Integration der Studienanfänger*innen in den Blick nehmen (Key u. Hill, 2018). Die verschiedenen Projekte setzen auf sehr unterschiedlichen Ebenen an. Allen Projekten gemeinsam ist aber eine enge Begleitung der Studierenden in der Übergangsphase zum Studium.

Mit der letzten Reakkreditierung wurde an der Hochschule Bremerhaven in den Bachelorstudiengängen Informatik und Wirtschaftsinformatik 2013 eine fachspezifische Studieneingangsphase (STEP) im Curriculum verankert. Ziel der Studieneingangsphase ist, das Berufsbild der Informatik und Wirtschaftsinformatik bei den Studierenden zu schärfen und das Zusammenspiel der Grundlagenveranstaltungen als Basis für die Informatik- und Wirtschaftsinformatikausbildung zu verdeutlichen. Die Idee, das Studium mit einem Projekt zu starten, gibt dabei den notwendigen Kontext für die vielfältigen Aufgaben in der Gestaltung und dem Einsatz von Softwaresystemen und lässt genügend Raum für Diskussionen über den vorhandenen Gestaltungsspielraum (Vosseberg, 2015). Mit den Projekten können die Studierenden erste Erfahrungen des eigenständigen, forschenden Lernens in Teams sammeln in einem eng begleiteten Lernkontext. Die Projekte fördern insbesondere die soziale Integration und den Austausch der Studierenden mit ihren vielfältigen Kompetenzen, die sie mit in ihr Studium einbringen. Um an dem Erfahrungshintergrund der Studierenden anzuknüpfen wurden in den ersten STEP-Jahren Analyseprojekte initiiert, da nicht davon ausgegangen werden kann, dass die Studierenden bereits umfangreiche Programmiererfahrungen mitbringen (Vosseberg u. a., 2015). Ähnliche Ansätze mit Projekten zum Studienstart werden auch an anderen Hochschulen verfolgt insbesondere mit dem

Fokus auf Studierbarkeit und das Fördern von sozialen Kompetenzen (vgl. (Dennert-Möller u. Garmann, 2016)).

Die Evaluation der ersten vier Durchläufe der Studieneingangsphase haben ergeben, dass die Startprojekte und die enge Begleitung der Projektteams durch studentische Tutor*innen und einem Coaching von Lehrenden die soziale Integration in beiden Studiengängen sehr gefördert haben. Die Verzahnung zu den Grundlagenfächern und die Angleichung von grundlegenden Fachkompetenzen war bislang jedoch nur bedingt gelungen. Gerade die Module Mathematik und Programmierung wurden nach wie vor als getrennt von der Studieneingangsphase wahrgenommen und die kontinuierliche Arbeitsweise aus den STEP-Projekten nicht übertragen. Ähnliche Effekte wie sie in der Auswertung der Umfrage zur Programmierausbildung von Axel Schmolitzky (Schmolitzky, 2017) beschrieben wurden, waren auch nach Einführung der Studieneingangsphase weiter zu beobachten. Nach wie vor sind die beiden Module Mathematik und Programmierung angstbesetzt, und die Modulprüfungen werden von einer überwiegenden Anzahl von Studierenden in spätere Semester geschoben.

Diese Erfahrungen haben uns dazu bewogen, das Lernsetting der Studieneingangsphase mit dem Wintersemester 2017/18 zu verändern. Gemäß dem Motto ”Informatik ist nicht nur Programmierung aber ohne Programmierung ist nichts Informatik” erarbeiten sich die Studierenden in einer Workshop-ähnlichen Lernumgebung erste Fertigkeiten in der Automatisierung von Abläufen und schleifen diese mit kleinen Übungen regelmäßig ein. Unterstützt werden die ca. 100 Studierenden durch Lehrende, die als Coaches zur Seite stehen, sowie durch studentische Tutorinnen und Tutoren. Das Einüben der Fertigkeiten bereitet sie auf die gestellte Projektaufgabe vor, die in dem überarbeiteten Konzept der Studieneingangsphase einen wesentlichen Anteil an Programmierung enthält. Im Folgenden wird das Lernsetting der Studieneingangsphase und insbesondere die Verzahnung mit den Modulen Mathematik und Software Engineering (SWE I) näher beschrieben.

Studieneingangsphase - Rahmenbedingungen

Die Studieneingangsphase ist an der Hochschule Bremerhaven durch den Modul Einführung in die Informatik bzw. Einführung in die Wirtschaftsinformatik im Curriculum der beiden Bachelorstudiengänge Informatik und Wirtschaftsinformatik verankert. Damit sind sowohl auf studentischer Seite ein Workload von 5 CP vorgesehen, als auch auf Seiten der Lehrenden 8 SWS Lehrleistung eingeplant. Im Rahmen eines Teamteachings betreuen 3-4 Lehrende die Studieneingangsphase, um damit auch die Verzahnung zu anderen Modulen im ersten Semester realisieren zu können. Zusätzlich wird den Studierenden über die enge Verzahnung mit den Aufgaben im Modul Software Engineering I weitere Zeit für das Einüben der für die STEP-Projekte notwendigen grundlegenden Fertigkeiten eingeräumt.

Die Projektorientierung ist nach wie vor eine zentrale Eigenschaft der Studieneingangsphase. Am ersten Studientag werden die Informatik- und Wirtschaftsinformatikstudierenden in 12 gemischte Teams mit jeweils 8- 10 Studierenden eingeteilt. Jedes Team wird durch eine*n Tutor*in über das ganze erste Semester begleitet und erhält einen Coach als erste Ansprechperson an die Seite. Zur Zeit betreut jeder der drei Coaches vier Teams, während die vier studentischen Tutor*innen je drei Teams betreuen und für diese ein STEP-Tutorium anbieten. Zusätzlich betreuen ältere Studierende die technische Infrastruktur der STEP-Teams (siehe unten). Außerdem wird für jedes Team zur Teamorganisation eine Gruppe im Rahmen des Lernmanagementsystems Ilias eingerichtet.

In den ersten Wochen werden neben Einzelaufgaben auch Teamaufgaben gestellt, um sukzessiv auf die Projektaufgabe aber auch auf das gesamte Studium vorzubereiten. Um eine Workshop-ähnliche Lernsituation zu erzeugen werden in einem großen Veranstaltungsraum 12 Gruppentische aufgebaut, an denen die Teams gemeinsam ihre Aufgaben bearbeiten und sich gegenseitig in den Einzelaufgaben unterstützen können. Der Ablauf der 3-4-stündigen Workshops ist geprägt durch einem Wechsel zwischen Inputs der Coaches, z.B. in Form einer kurzen Einführung in ein Thema oder von kleinen Live-Coding-Einheiten, dem selbständigen Üben von Fertigkeiten und der gemeinsamen Bearbeitung der Projektaufgabe im Team. Die Sitzungen sind geprägt durch eine sehr arbeitsintensive Atmosphäre, in der alle Studierenden konzentriert mitarbeiten. Daneben werden sie mit kleinen Wochenaufgaben angehalten, während der Woche regelmäßig - am besten täglich - sich mit der technischen Infrastruktur auseinanderzusetzen, um die einfachen Fertigkeiten in der Automatisierung einzuschleifen.

Im Rahmen einer Portfolio-Prüfung für den Modul Einführung in die Informatik bzw. Einführung in die Wirtschaftsinformatik müssen die Studierenden die gestellten Aufgaben bearbeiten und jede Woche einen Eintrag in ihrem Reflektionsblog in das Lernmanagementsystem einstellen. Am Ende des Semesters stellen die Projektteams ihre Projektergebnisse am Tag der Informatik mit einem Plakat vor. Über die regelmäßige Bearbeitung der gestellten Einzel- und Teamaufgaben insbesondere aber über die wöchentlichen Blogeinträge erhalten die Coaches einen guten Einblick über den Lernfortschritt der Studierenden und können bei Problemen sofort gegensteuern. Nach anfänglichen Schwierigkeiten werden die Blogeinträge regelmäßig geführt und sind damit eine wertvolle Informationsquelle für die Evaluation der Studieneingangsphase.

Um eine organisatorische Verzahnung zwischen den verschiedenen Modulen im ersten Semester zu unter-

stützen, werden die Teams als Lerngruppen in allen Modulen genutzt und in der Gruppenaufteilung im Stundenplan berücksichtigt. Zusätzlich übernehmen Lehrende aus dem ersten Semester eine Gruppe (3 Teams) im Rahmen des Programmierlabors, damit die Verzahnung zum Programmierenlernen sichtbarer wird. Die Programmierlabore werden mit zusätzlichen Tutor*innen unterstützt. Somit haben wir eine sehr enge Begleitung der Erstsemesterstudierenden und sie lernen sehr frühzeitig viele Kolleg*innen aus dem Informatikbereich mit ihrer eigenen Vielfalt kennen.

Inhalte der Studieneingangsphase

Im Zentrum der Studieneingangsphase stehen die Projekte, die einen Anwendungskontext bieten und zum Erlernen von grundlegendem Handwerkzeug der Informatik motivieren, das die Studierenden für die Realisierung der Projekte benötigen. Dieses Handwerkszeug wird sie auch durch ihr gesamtes Studium begleiten. Um die Studieneingangsphase abzurunden und einen praktischen Einblick in Berufsbilder der Informatik und Wirtschaftsinformatik zu erhalten, besuchen alle Teams an einem Tag jeweils ein Unternehmen aus der Region.

Für den Unternehmensbesuch werden 12 ganz unterschiedliche Unternehmen ausgewählt, von klassischen Softwareentwicklungshäusern, spezialisierten Unternehmen beispielsweise aus der Logistik- oder Lebensmittelbranche bis hin zu Beratungsunternehmen oder IT-Abteilungen von Konzernen. Die Teams müssen für den Unternehmensbesuch Fragen vorbereiten, die insbesondere an ihren bisherigen Erfahrungen mit der neu erlernten Infrastruktur und den Arbeitsweisen anknüpfen. Nach dem Unternehmensbesuch werden die Erfahrungen in Form eines World-Cafes mit allen Studierenden geteilt. Damit wird die Vielfalt der beruflichen Möglichkeiten sichtbar. Außerdem wird diskutiert, welche Grundlagen die Studierenden in ihrem Studium brauchen, um in Zukunft solche oder ähnliche Jobs bewältigen zu können.

Projekte 2017/2018 und 2018/2019

Die Projektaufgabe der Studieneingangsphase 2017/2018 bestand darin, Modellboote, die jeweils mit einem Raspberry Pi ausgestattet und steuerbar sind, über einen Web-Browser per WLAN zu lenken und zu beschleunigen bzw. zu verlangsamen. Zur Vorbereitung dieser Projekte hatten ältere Studierende die Modellboote nach einer Konstruktionszeichnung des Deutschen Schifffahrtsmuseums Bremerhaven (DSM) für den Druck per 3D-Drucker aufbereitet. Für jedes der 12 STEP-Teams haben sie ein Boot gedruckt und mit Motor, Steuerruder und Raspberry Pi versehen (siehe Abbildung 1).

Eine Besonderheit bei dieser Aufgabe bestand darin, dass der Original-Schlepper, der als Vorlage für die Modellboote diente, seinen Liegeplatz direkt vor

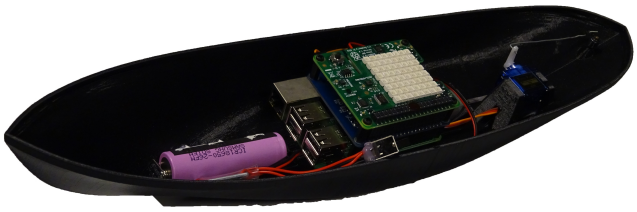


Abbildung 1: Modellboot mit Raspberry Pi

dem STEP-Veranstaltungsraum, dem ehemaligen Fährhaus, hat und von dort aus oft zu sehen war (siehe Abbildung 2).

Abbildung 2: Der Schlepper Tide

Als Hochschule am Meer haben wir auch in der Studieneingangsphase 2018/2019 ein maritimes Thema gewählt. Dieses Mal wurde ein Raspberry Pi mit einem DVB-T-Empfänger verbunden und oben auf dem Fährhaus stationiert. Über diesen Rechner werden permanent AIS-Signale aller Schiffe in einem Umkreis von bis zu 50 km empfangen, dekodiert und in einem bestimmten Format als Datenstrom in unserer Informatik-Infrastruktur zur Verfügung gestellt. Das AIS ist das Automatische Schiffsidentifizierungssystem (engl. Automatic Identification System) und dient dem Austausch von Navigations- und Schiffsdaten zur Verbesserung der Sicherheit im Schiffsverkehr (WSV, 2018).

Die Projektaufgabe besteht darin, aus diesem Datenstrom Angaben zu ausgewählten Schiffen zu filtern und auf einer Website per HTML und SVG zu veranschaulichen. Zum Beispiel können die Positionen und Bewegungen von Schiffen relativ zum Fährhaus und auch auf einer Karte angezeigt werden. Dazu können Informationen wie Schiffsname oder Zielort gegeben werden. Die Entscheidung über die Darstellungsart der Daten und die Anzeige zusätzlicher Informationen bleibt den STEP-Teams überlassen. Daraus ergeben sich Webseiten, bei denen eher Informationen über die Schiffe im Vordergrund stehen, und solche, die die

Möglichkeiten dynamischer Webseiten in der gegebenen Infrastruktur ausloten.

Anhand der beschriebenen Aufgabenstellungen werden grundlegende Kenntnisse von Bash-Shell-Skripten und HTML eingeübt sowie ein Verständnis für webbasierende Client-Server-Architekturen vermittelt.

Handwerkszeug der Informatik und Wirtschaftsinformatik

An der Hochschule Bremerhaven wurde innerhalb der vergangenen zwei Jahre wieder eine klassische Linux-basierte Infrastruktur aufgebaut, wie sie in den 80er und 90er Jahren an vielen Universitäten und Hochschulen in der Informatik zum Standard gehörte. Damit wird der aktuellen Entwicklung Rechnung getragen, in der Webtechnolgien und insgesamt Server-orientierte Anwendungen einen massiven Aufschwung erleben.

So besteht die gemeinhin als *Cloud* bezeichnete technolgische Basis moderner Anwendungen letztlich aus einer Menge an Servern mit Rechen- und Speicherkapazität. Anwendungen bestehen dort aus Programmen, die typischerweise unter Linux laufen. Selbst in der Microsoft-eigenen Azure-Cloud laufen zunehmend Anwendungen unter Linux:

> *Today, Scott Guthrie, Microsoft's executive vice president of the cloud and enterprise group, said in an interview, "it's about half now, but it varies on the day because a lot of these workloads are elastic, but sometimes slightly over half of Azure VMs are Linux."*
>
> (Steven J. Vaughan-Nichols)

Zudem ist das *InternetOfThings* und große Teile dessen, was unter *Industrie 4.0* verstanden wird, auf unixoiden System – speziell Linux – aufgebaut.

Folglich müssen heute Studierende wieder darauf vorbereitet werden, sich in einer solchen Umgebung sicher bewegen zu können.

Unsere Umgebung besteht aus mittlerweile mehreren Arbeitsservern, auf denen sich die Studierenden innerhalb der Hochschule ebenso wie von außerhalb per *ssh* einloggen können. Sie finden dort ein Home-Verzeichnis für Ihre Arbeiten vor und lernen von Beginn an, mit dem Editor *vim* Ihre Aufgaben zu erledigen. Zudem haben sie dort ein Webverzeichnis, in dem sie ihre persönliche Webseite mit HTML und CSS gestalten können und sollen. Die Infrastruktur bietet jedem Studierenden und jedem Lehrenden außerdem Zugang zu einer eigenen Datenbank, zu PHP, Java und einem Git-Server.

Da wir neben Java und der Bash im ersten Semester nicht noch eine weitere Programmiersprache mit PHP einführen wollten, trotzdem aber mit dynamisch erzeugen Webseiten bereits Automatisierung in das Zentrum der Vorbereitung auf die kommenden Semester stellen wollten, haben wir für das Wintersemester 2018/2019 eine Docker-basierte Umgebung aufgesetzt (Merkel, 2014). Darin erhält jeder und jede Studierende einen eigenen gut ausgestatteten Container mit ssh- und Webserver, auf dem sie sich wiederum einloggen und mit cgi-Skripten dynamisch Seiten erzeugen können. Java ist dort ebenfalls installiert, so dass sie ihre beginnenden Java-Kenntnisse frühzeitig anwenden können, ohne mit der Komplexität von Enterprise-Java (Servlets etc.) oder GUI-Programmierung (AWT/Swing) schon konfrontiert zu werden.

Auf den Arbeitsservern sowie in den Containern ist eine durchaus typische Server- und Arbeitsumgebung installiert. Aufgrund der Gegebenheit, von Beginn an nur in der Kommandozeile zu arbeiten, um alle Arbeitsschritte wiederholbar und automatisierbar durchführen zu können, trotzdem aber die Studierenden nicht auf die für sie zunächst unmodern und unhandlich wirkende reine Textausgabe zu beschränken, wurde ein spezifischer Arbeitsfluss etabliert: Wo immer sie etwas mit einem selbst geschriebenen Programm erzeugen, nutzen sie das Webverzeichnis, um sich das Ergebnis im Browser anzeigen lassen zu können und gegebenenfalls auch Freunden und Familie frühzeitig zu zeigen, was sie im Studium tun. So lernen sie früh das Werkzeug *gnuplot* kennen, um Graphen visuell ansprechend als PDF oder SVG zu erzeugen und sie lernen LaTeX, um Dokumente so zu generieren, dass sie zum einen heutigen ästhetischen Ansprüchen genügen als auch die sorgfältige Trennung von Struktur und Darstellung verdeutlichen. Da sie frühzeitig HTML von Hand und ohne all zu mächtige Hilfsmittel zu schreiben angehalten werden, können sie systematisch Listen und Tabellen mit Skripten oder mit Java erzeugen.

Der grundsätzliche Arbeitsrhythmus ist also: *Erstelle ein Programm, das etwas produziert, das im Web angezeigt werden kann - sei es ein HTML-Dokument, eine PDF-Datei, ein Bild (PNG) oder eine Grafik (SVG/PDF)*. Wie in dem Abschnitt zum *Workshop Videoerstellung* dargestellt, gehen wir dabei soweit, dass aus einer Menge von generierten Grafiken automatisiert eine Menge von Bildern und daraus dann ebenfalls automatisiert ein Video erstellt werden kann. Die Werkzeuge dafür (*inkscape* und *ffmpeg*) gehören daher auch zu unserer Standardumgebung.

Verzahnung von Modulen

Um Studierenden einen Zugang und ein entsprechendes Verständnis für grundlegende Konzepte der Informatik zu ermöglichen, verfolgen wir in der Studieneingangsphase den Ansatz, verschiedene Fachmodule derart miteinander zu verzahnen, dass spezifische Themen aus jeweils einem der Module auch in anderen Modulen aufgegriffen und behandelt werden. Eine solche Verzahnung ermöglicht es, Themen aus verschiedenen Perspektiven und anhand von verschiedenen Beispielen zu betrachten. Die Hoffnung dabei ist, dass mehr Studierende zeitnah beim Stoff der Vorlesun-

gen mitgenommen werden und verschiedene Lerntypen die Chance haben, den Stoff zu erfassen. Für die Verzahnung ist es wichtig, dass sich die beteiligten Lehrenden der jeweiligen Module (mindestens der Module STEP, Programmieren, Mathematik, Software Engineering) abstimmen, in welchen Wochen voraussichtlich welche Themen behandelt werden. Es geht dann nicht darum, auf dem Vorwissen aus den jeweils anderen Veranstaltungen aufzusetzen, sondern das Thema mit dem Ansatz und aus der Perspektive der jeweiligen Veranstaltung einzuführen und dabei ggf. Beziehungen zu den anderen Veranstaltungen herzustellen. Dieses Konzept trägt auch den pädagogischen Erkenntnissen Rechnung, dass neues Wissen nachhaltiger erworben werden kann, wenn es in einen breiten Kontext vorhandenen Wissens eingeordnet werden kann:

> *"As just explored, a wide range of converging evidence stresses the fundamental importance of the learning edge, the boundaries of existing knowledge which form the context into which newly learned concepts (information, understanding and skills) must be integrated. This fact is widely understood and accepted, and the basis of many sound pedagogical practices."*
>
> (Robins, 2010, S.66)

Unser Curriculum sieht insgesamt drei Module für Software Engineering (SWE) vor: Im ersten Semester geht es um Modellierung im Allgemeinen und UML im Besonderen. Im zweiten Semester wenden die Studierenden die gelernten Methoden in einem konkreten Kundenprojekt ihrer Wahl an, während im dritten Semester anhand von Fallbeispielen ein besonderer Fokus auf Software-Architekturen und Qualitätssicherung gelegt wird. Die enge Verzahnung mit Inhalten des STEP-Projektes bietet in SWE I die Chance, Modellierungsbeispiele direkt auf die sehr praxisbezogenen Aufgabenstellungen der STEP-Projekte zu beziehen. Insbesondere die Erstellung von UML-Aktivitätsdiagrammen zu konkreten bash-Skripten aus den STEP-Projekten führt einerseits zu einem besseren Verständnis für den Sinn der Modellierung. Andererseits erhalten die Studierenden dadurch einen anderen, visuellen Blick auf den Quellcode, der vielen einen weiteren Zugang zum Verstehen der Programme und Programmierkonzepte ermöglicht.

Im Folgenden skizzieren wir einige Beispiele zur Verzahnung von Modulen aus der Studieneingangsphase unserer Bachelorstudiengänge Informatik und Wirtschaftsinformatik in den Wintersemestern 2017/2018 und 2018/2019.

Modellbildung

Die Aufgabenstellung des STEP-Projektes 2017/2018 eignete sich besonders gut zur Erläuterung einiger Grundprinzipien der Modellierung im Allgemeinen, die in SWE I vermittelt werden. Zum Original, dem Schlepper Tide (siehe Abbildung 2), der seinen Liegeplatz direkt vor dem Fährhaus hat, gibt es mehrere Modelle: Zum einen diente die Konstruktionszeichnung (siehe Abbildung 3) als präskriptives Modell sowohl für den Original-Schlepper als auch für den 3D-Druck der Modellboote.

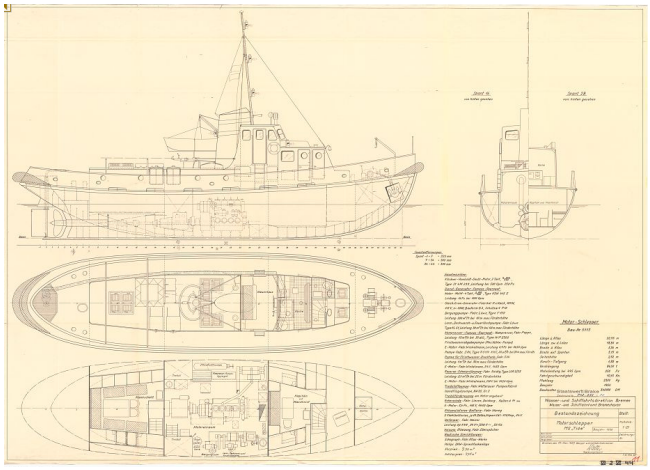


Abbildung 3: Konstruktionszeichnung der Tide (DigiPEER)

Zum anderen ist das ausgedruckte STEP-Boot (siehe Abbildung 1) ein Modell, das einige wenige Merkmale des Originals abbildet. Abbildung, Verkürzung und Pragmatismus, die von Stachowiak (Stachowiak, 1973) benannten typischen Merkmale von Modellen, werden an diesem Beispiel greifbar.

Neben solchen allgemeinen Aspekten der Modellierung erfahren die Studierenden ganz konkret den Nutzen der Modellierung beim Software Engineering, wenn sie in SWE I die Aufgabe erhalten, das Konzept für die im STEP-Projekt zu entwickelnde Anwendung zunächst mittels UML-Anwendungsfalldiagramm zu entwerfen und später einzelne Abläufe per UML-Aktivitätsdiagramm zu konkretisieren. In den STEP-Projekten werden keine festen Vorgaben zur erwarteten Funktionalität gemacht, sondern vor allem ein Rahmen abgesteckt. Die Idee für das Endprodukt wird von den Teams jeweils selbst entwickelt. Anwendungsfalldiagramme sind dann eine gute Methode für die Verständigung im Team über das zu entwickelnde System.

Kontrollstrukturen, Aktivitätsdiagramm und mathematische Berechnungen

In der STEP-Veranstaltung werden stets auch Programmierprinzipien aus der Java-Programmierveranstaltung aufgegriffen und deren Umsetzung in der Bash gezeigt. Auf diese Weise wird die Programmiersprachen-übergreifende Bedeutung von Konstrukten wie bedingten Anweisungen und Schleifen unterstrichen. In SWE erhalten die Studierenden durch die Modellierung dieser Konstrukte in Aktivitätsdiagrammen auch einen visuellen Zugang. Ein anschauliches Beispiel ist die Modellierung der

Erhöhung der Geschwindigkeit der STEP-Boote aus dem STEP-Projekt 2017/2018. Die Boote konnten per Browser durch den Aufruf entsprechender CGI-Skripte gesteuert werden, die auf einem Webserver, dem Raspberry Pi der Boote, ausgeführt wurden. Ein Skript diente z.B. dazu, die Boote zu beschleunigen, bis sie sich zu sehr in Schieflage neigten. In SWE I sollte dies mittels UML-Aktivitätsdiagrammen mit 'Schwimmbahnen' modelliert werden. Eine mögliche Lösung dazu findet sich in Abbildung 4. Der zyklische Verlauf einer Schleife kann im Aktivitätsdiagramm gut veranschaulicht werden.

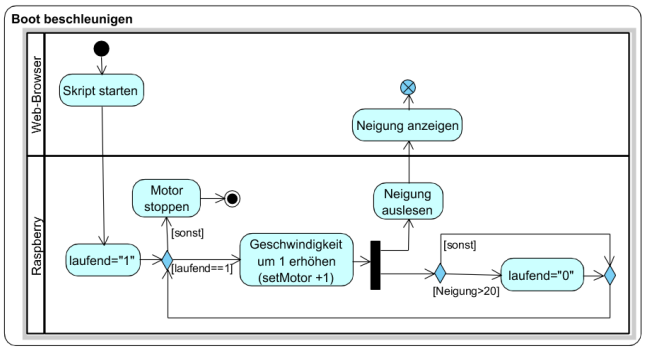


Abbildung 4: Aktivitätsdiagramm Boot beschleunigen

Die Nutzung von Schleifen wurde auch zum Beispiel anhand der Dezimal-Binär-Umwandlung mit Modulo-Berechnung verdeutlicht. Mit diesem Beispiel wird sowohl ein Bogen zur Mathematik-Veranstaltung geschlagen, als auch das bash-scripting einer Schleife gezeigt (siehe Abbildung 5). Zudem kommen dafür in den Aktivitätsdiagrammen Notationselemente wie Verzweigung und Zusammenführung zum Einsatz.

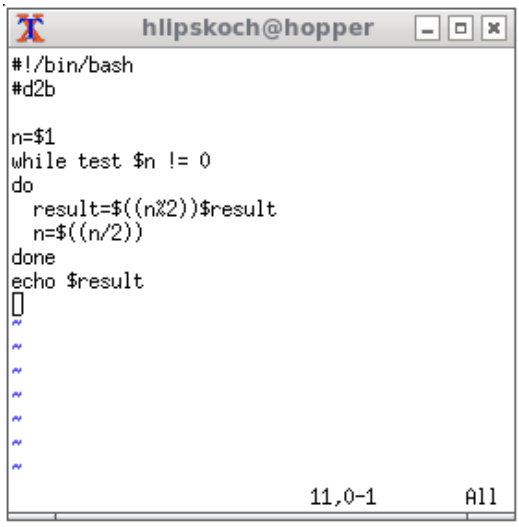


Abbildung 5: Bash-Skript Dezimal-Binär-Umwandlung

Verzahnung mit der Mathematik am Beispiel RSA

Im Modul Mathematik werden die Begriffe Binärzahlen, Dividieren mit Rest und Kombinatorik mit der Fakultät in jeweils eigenen Veranstaltungseinheiten eingeführt. Binärzahlen als Grundlage der digitalen Computertechnik und das Dividieren mit Rest, welches ein Beispiel für die Endlichkeit mathematischer Berechnungen im Computer darstellt, erfahren durch die Verzahnung mit STEP direkte praktische Übungen. Diese Übungen wären sonst im Modul Mathematik aufgrund der erforderlichen Themenbreite für Unix, Shell-Skripte etc. nicht zu leisten.

Insbesondere kann so das RSA-Verschlüsselungsverfahren (Rivest u. a., 1977) praktisch ausprobiert werden: In der Mathematik lernen die Studierenden den Algorithmus und die mathematische Grundlage. Die Basis bilden zwei sehr große Primzahlen und dann werden nacheinander mehrere Zahlen daraus berechnet. Dabei wird auch das sogenannte Multiplikative Inverse benötigt. Hierzu lernen die Studierenden ebenfalls in der Mathematik den Euklidischen Algorithmus (Knuth, 1998) kennen und anwenden. Für die korrespondierende STEP-Aufgabe ist ein Programm, welches den Euklidischen Algorithmus ausführt und die einzelnen Schritte auf der Konsole darstellt, gegeben.

In Abbildung 6 ist die Ausgabe der Berechnung zu sehen. In der Zeile für $i = 5$ ist der Wert von r gerade 1, was bedeutet, dass die Zahlen 310 und 617 keine gemeinsamen Teiler haben. Deswegen gibt es das Multiplikative Inverse bezüglich modulo 617. Es steht in der gleichen Zeile in der Alpha-Spalte. Da die Zahl negativ ist, müssen die Studierenden zusätzlich 617 addieren:

$$(310 \cdot (-205 + 617)) \mod 617 = 1$$

Diese Teilaufgabe ist nicht ohne die mathematischen Kenntnisse zu lösen.

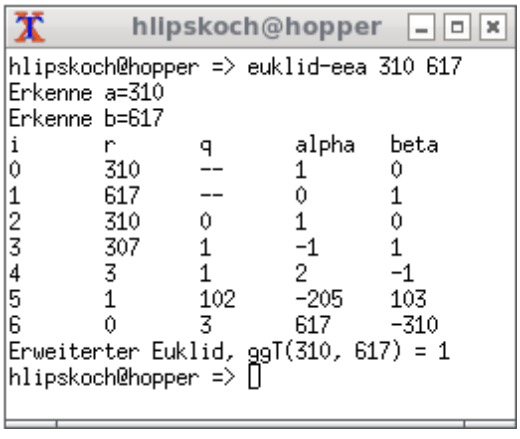


Abbildung 6: Erweiterter Euklidischer Algorithmus berechnet für die Zahlen 310 und 617

Die gesamte STEP-Aufgabe besteht daraus, dass die Studierenden zwei große Primzahlen finden und die verschiedenen weiteren benötigten Zahlen für RSA vermittels einfacher mathematischer Formeln (Multiplikationen) berechnen. Wie sie die Primzahlen finden, ist ihnen überlassen. Durch Shell-Skript-Aufruf

bestimmen sie das Multiplikative Inverse, in dem das gegebene Programm aufgerufen und aus der Ausgabe der entsprechende Wert herausgefiltert wird.

Danach besteht die Aufgabe daraus, mit den hergestellten Zahlen, das RSA-Verfahren anzuwenden: Das bedeutet wieder ein Skript zu schreiben, welches in der Lage ist Potenzen und Modulo zu berechnen. Am Schluss haben die Studierenden alle nötigen Werkzeuge hergestellt, um sich gegenseitig verschlüsselte Nachrichten zuzusenden und zu entschlüsseln. Die vier Hauptaufgaben eines Verschlüsselungsverfahrens finden Anwendung: Verschlüsselung, Entschlüsselung, Signierung, Verifizierung.

Workshop Videoerstellung

Die Vermittlung der Kompetenz, selbst Visualierungen zu generieren, ist ein weiteres Thema im Rahmen der Step-Veranstaltung, auf das wir im Folgenden eingehen.

Videos werden von heutigen Studierenden gern und viel zum Lernen genutzt. Jedoch sind fremdgeschaffene Lernvideos zunächst einmal ein passives Medium und bieten Ihrer Natur nach eben keine Möglichkeit, den dargestellten Sachverhalt mit eigenen Parametern zu variieren oder mit eigenen Ideen anzureichern.

Das hier vorgestellte Instrument *selbsterstellte Videos zur Visualisierung komplexer Vorgänge* knüpft an die konstruktionistischen Ideen von Seymore Papert an:

> *Slowly I began to formulate what I still consider the fundamental fact about learning: Anything is easy if you can assimilate it to your collection of models. If you can't, anything can be painfully difficult.*
>
> (Papert, 1980, S. iiv)

Große Teile der Ideen von Papert sind explizit auf den Umgang von Schüler*innen mit Computern ausgerichtet, um sowohl das Programmieren als solches zu erlernen als auch die Lust am mentalen Modellieren und Prüfen der Modelle zu fördern. Demzufolge sind auch in dieser Tradition fortgeführte Programmierumgebungen (von Logo bis Scratch) nicht auf generische Programmiersprachen ausgerichtet und bisweilen für Studierende zu verspielt.

In unserem Ansatz verfolgen wir das Ziel, dass die Studierenden eine Grundmenge an Elementen an die Hand bekommen, mit der sie sich aus jeder Programmiersprache heraus selbst Visualisierungen höherer und abstrakterer Konzepte erstellen können.

Scalable Vector Graphics

Das Format SVG (Scalable Vector Graphics) hat sich in den vergangenen Jahren zu einem Standardformat für Vektorgrafiken entwickelt. Jeder einigermaßen moderne Browser kann SVG-Grafiken nativ (also ohne Plugins) darstellen. SVG ist ein reines, auf XML basierendes Textformat - grafische Primitive wie Kreis, Rechteck, Linienzug oder Text können direkt im Editor eingegeben werden:

```
<svg width='400' height='200'
  xmlns='http://www.w3.org/2000/svg'>
  <rect x='0' y='0'
        width='400' height='200'
        stroke-width='1'
        stroke='black' fill='whitesmoke'/>
  <circle cx='100' cy='100' r='50'
          stroke='black' fill='gray' />
  <text x='10' y='30'>
    Ohne Programmieren ist nichts
    Informatik</text>
</svg>
```

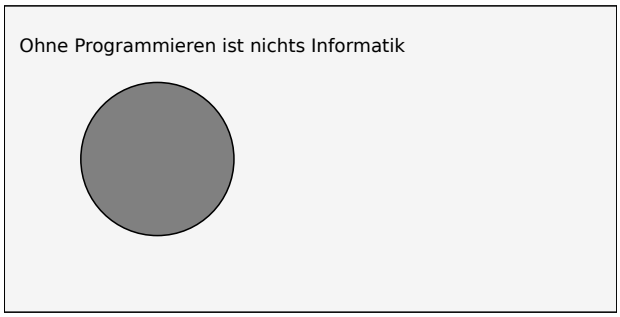


Abbildung 7: SVG Ausgabe

Es genügt zum Erstellen einer Grafik hier also zunächst die Fertigkeit zur Bedienung eines Editors und die Beherrschung einiger weniger Unix-Kommandos wie *cp* für das Kopieren der Datei in das Webverzeichnis. Beim Editieren ist immer auch ein Browser geöffnet, in dem das Ergebnis sehr schnell geprüft werden kann. Da die Studierenden zu dem Zeitpunkt, da wir SVG eingeführt haben, bereits HTML-Kenntnisse besitzen, ist die Syntax von SVG nicht mehr all zu fremd - die Tatsache, dass Baumstrukturen mit ineinander geschachtelten Tags in der speziellen Notation mit spitzen Klammern konstruiert werden, ist folglich keine große Hürde mehr. Das schnell wechselnde Arbeiten zwischen Editor und Browser ist ebenfalls eingeübt und stellt keine Probleme mehr dar.

Einheitskreis

Als bedeutend größere Schwierigkeit stellt sich heraus, dass die Studierenden zwar mit den Worten Sinus, Cosinus, Pythagoras und Einheitskreis vertraut waren, jedoch die Zusammenhänge mit Koordinatensystemen ihnen zu einem erheblichen Teil so fremd waren, dass sie nicht - oft auch kaum nach ausführlicher Hilfe in Tutorien - in der Lage waren, einen Kreis aus 18 Kreisen zu konstruieren. Der Stoff aus dem Mathematikunterricht spätestens der 10. Klasse war nicht präsent - geschweige denn zu einem Teil der inneren Modellbildung geworden. Für einen Studiengang, der heutzutage so stark auf Visualisierungen und Grafik im Allgemeinen aufbaut, stellt das eine

ernstzunehmende Problematik dar, der zu begegnen ist; zumal nicht wenige der Studierenden selbst sehr visuell geprägt sind und das Programmieren von Computerspielen recht weit oben auf der Wunschliste der eigenen Befähigungen steht. In Zeiten, in denen 3D-Visualisierungen und -Animationen für jeden programmierwilligen Menschen erschwinglich und machbar ohne Spezialausrüstung sind, wird man die Studierenden gerade zu diesem so zukunftsträchtigen und spannenden Feld so nicht hinführen können: Kein API oder Framework nimmt einem die Notwendigkeit ab, mit drei ausgestreckten Fingern vor sich Rotationen im dreidimensionalen Raum in Cosinus und Sinus denken zu müssen.

Grundannahme

Wer

- bereits einfache Skripte schreiben kann, die Texte ausgeben,
- Ausgaben auf der Kommandozeile ganz problemlos in eine Textdatei umleiten kann,
- Schleifen und Bedingungen ausdrücken kann,
- versteht, dass Programmaufrufe unter Unix immer sowohl in der Shell ausführbar sind als auch genau so in einem Skript mit Kontrollstrukturen zur Automatisierung stehen können,
- ein Kommandozeilenprogramm zur Verfügung hat, das SVG-Grafiken in PNG-Bilder konvertieren kann (inkscape) und
- ein Kommandozeilenprogramm zur Verfügung hat, das eine Menge von PNG-Bildern in ein digitales Video konvertiert (ffmpeg),

kann sich mit wenig Aufwand ein eigenes Video programmieren, in dem der Zusammenhang von Sinus, Cosinus und Einheitskreis dargestellt wird.

Im folgenden Abschnitt zeigen wir die konkrete Lerneinheit, die in Form eines Workshops an einem Vormittag den Studierenden etwa in der Mitte des Semesters gegeben wurde, nachdem in der Vorwoche klar wurde, dass Trigonometrie noch einmal vertieft werden könnte.

Ein Workshop zum Einheitskreis

1. *Schreiben Sie ein Skript, das eine SVG-Datei der Breite 700 Einheiten und der Höhe 300 Einheiten ausgibt, in dem ein Rechteck derselben Größe an dem Punkt (0,0) positioniert wird.*

2. *Fügen Sie einen Kreis mit dem Radius 100 Einheiten an der Position (150,150) ein und zeichnen Sie ein Koordinatenkreuz ein, das durch eben diesen Mittelpunkt des Kreises geht. Für die folgenden Überlegungen gilt dieser Punkt als Nullpunkt in dem kartesischen Koordinatensystem und 100 graphische Einheiten entsprechen einer Einheit in diesem System.*

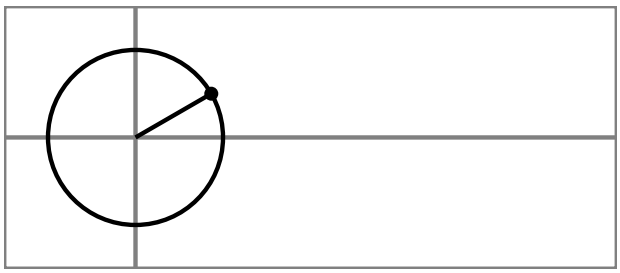


Abbildung 8: SVG Ausgabe für die Längen im Einheitskreis bei 30 Grad

3. *Variieren Sie Ihr Skript so, dass es als Argument eine Zahl in Graden von 0 bis 360 entgegennimmt und zeichnen Sie eine Linie von dem Mittelpunkt des Kreises bis zum Rand des Kreises bei der übergebenen Gradzahl. Fügen Sie dort noch einen Kreis mit dem Radius 8 ein. Berechnen Sie die Koordinaten mit bc.*

   $rad = \frac{degrees \cdot \pi}{180}$

   $x = cx + \cos(rad) \cdot r$

   $y = height - (cy + \sin(rad) \cdot r)$

   ```
   rad=$(echo ''($deg * 4*a(1))/180''|bc -l)
   x=$(echo ''$cx + s($rad)*$r''|bc -l)
   y=$(echo ''$h-($cy + c($rad)*$r)''|bc -l)
   ```

4. *Rufen Sie in einer Schleife mit i von 0 bis 360 Ihr Skript mit i als Argument auf und erzeugen Sie dadurch 361 SVG-Dateien der Form kreis$num.svg, wobei sich $num aus $\$i + 1000$ berechnet. Erzeugen Sie aus den SVG-Dateien ebenfalls in der Schleife jeweils eine PNG-Datei:*

   ```
   inkscape -z --export-png=kreis$num.png \
   kreis$num.svg
   ```

5. *Benutzen Sie ffmpeg nun, um aus den 361 Dateien eine Videodatei zu erzeugen:*

   ```
   ffmpeg -y -i kreis1%03d.png \
  -pix_fmt yuv420p kreis.mp4
   ```

   Betrachten Sie das Video in Ihrem Browser.

6. *Verändern Sie nun Ihr Skript so, dass zusätzlich zu der Linie, die vom Mittelpunkt zum Kreisrand gezogen wird, noch die Linie vom Kreismittelpunkt zu der berechneten X-Koordinate aber auf der gleichen Höhe wie der Kreismittelpunkt in blau eingezeichnet wird. Nun zeichnen Sie noch eine Linie von diesem Punkt zu dem berechneten Punkt auf dem Kreisrand in rot. Erzeugen Sie nun wieder ein Video und betrachten Sie den Verlauf des rechtwinkligen Dreiecks.*

7. *Verändern Sie in einem vorerst letzten Schritt noch einmal Ihr Skript so, dass – wieder in einer Schleife – alle Werte von 0 bis zur aktuellen Gradzahl für Cosinus (x-Achse) und Sinus (y-Achse) berechnet und neben dem Kreis als einzelne kleine Kreise mit einem Radius von 5 Einheiten aufgetragen werden.*

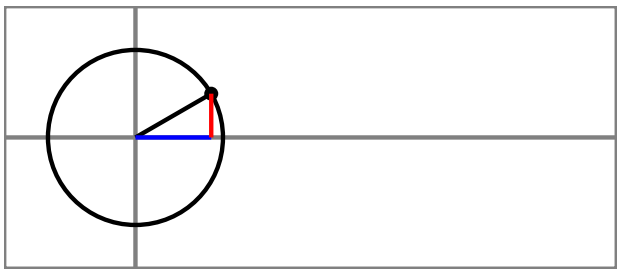


Abbildung 9: SVG Ausgabe für die Längen im Einheitskreis zur Veranschaulichung von Sinus und Cosinus bei 30 Grad

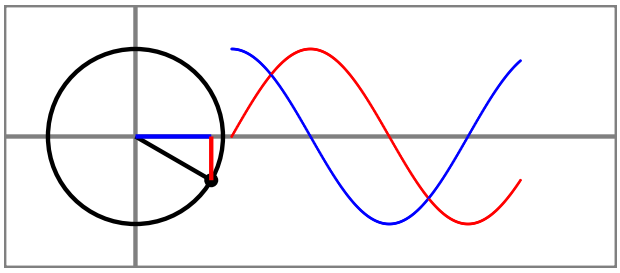


Abbildung 10: SVG Ausgabe für die Längen im Einheitskreis zur Veranschaulichung von Sinus und Cosinus bei 330 Grad mit Sinus- und Cosinus-Kurve

8. *Betten Sie die das Video in Ihre HTML-Datei nun so ein, dass es in einer Endlosschleife abläuft und betrachten Sie Ihr Werk einige Minuten. Versuchen Sie so das Verhältnis von Sinus und Cosinus im Einheitskreis als Längen in einem Koordinatensystem zu begreifen.*

Kontextualisierung

SVG als Visualisierungswerkzeug schult die Studierenden von Beginn an darin, in Schnittstellen zu denken. Statt spezieller APIs oder sogar Programmiersprachen wird in guter Unix-Tradition als Interface reiner Text genutzt, der über Pipes und Ausgabeumlenkung in Dateien gelenkt wird. Zusammen mit der Möglichkeit, in Schleifen viele, systematisch benannte Dateien zu erzeugen und mit einem einzigen Programmaufruf in ein Video zu konvertieren, lassen sich selbständig komplexe Sachverhalte durchdenken. Das oben beschriebene Konzept führen wir bei uns in der Studieneingangsphase etwa in der Mitte des Semesters ein. Das Dateisystem, Schleifen, Bedingungen und einfache Skripte sind den Studierenden bis dahin ebenso wie HTML und SVG hinreichend nahe gebracht worden. Dass das Beispiel Sinus und Cosinus am Einheitskreis behandelt wird, ist lediglich der konkreten Situation geschuldet, dass klar wurde, dass zumindest in dieser Kohorte diesbezüglich Nachholbedarf bestand. Es könnte sich ebenso gut um einfache Sortieralgorithmen, das Durchlaufen von Arrays oder Algorithmen auf Graphen handeln.

Ebenfalls lässt sich das Konzept auf die Programmierung mit Java anwenden - allerdings nur in Verbindung mit der Shell. Das Öffnen und Schließen und Beschreiben von Dateien in Java erfordert ein komplexes Geflecht aus Objekten und Klassen, die zu Beginn des Programmierenlernens noch zu fremd sind. Mit *System.out.println* lässt sich jedoch für jeden Programmaufruf ein Datenstrom beschreiben - und so wird in einer Shell-Schleife jeweils ein Java-Programm aufgerufen, das wiederum SVG hinausschreibt.

Nach dem Workshop haben wir den Studierenden als Wochenaufgabe gestellt, mit den gleichen Konzepten (SVG, Bash und ffmpeg) *irgendeine* Animation zu erstellen - möglicherweise eine, die etwas aus dem bisherigen Studium veranschaulicht. Dabei sind sehr beeindruckende Ergebnisse entstanden: mehrfach wurde ein Sortieralgorithmus visualisert und die *Türme von Hanoi*, die in der Programmiervorlesung behandelt worden waren, fanden sich ebenfalls mehrfach wieder. Dass in fast allen anderen in irgendeiner Form mit den Kreisfunktionen experimentiert wurde (von Analoguhren bis zu Lissajous-Figuren) war angesichts der oben beschriebenen Ausgangssituation ein besonders erfreuliches Ergebnis.

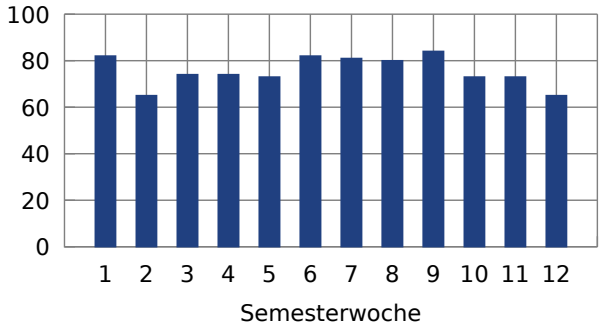
Fazit

Die mittlerweile sechsjährigen Erfahrungen mit der Studieneingangsphase an der Hochschule Bremerhaven haben gezeigt, dass ein begleiteter, projektorientierter Studienstart die soziale Integration der Studierenden fördert und in den meisten Teams die Zusammenarbeit auch nach anfänglichen Schwierigkeiten sehr gut funktioniert. Die Teams kommen zu guten bis sehr guten Projektergebnissen und die Einzelnen zeigen deutliche Fortschritte in ihrer Lernentwicklung. Mit der Neuausrichtung der Projekte von reinen Analyseaufgaben zu Programmieraufgaben kann eine bessere Verzahnung der Grundlagenfächer hergestellt werden. Somit werden die Zusammenhänge für die Studierenden erfahrbarer.

Ob alle angestrebten Ziele auch tatsächlich erreicht werden, bedarf noch entsprechender Evaluationen. Bisher gründen unsere Aussagen dazu auf eigenen Eindrücken sowie auf den STEP-Blogs der Studierenden. Wie erwähnt wird von den Studierenden erwartet, dass sie in ihrem Teamordner auf der ILIAS-Lernplattform einen wöchentlichen, persönlichen Blogeintrag schreiben, in dem kurz zusammengefasst wird, was in allen Lehrveranstaltungen des ersten Semesters gelernt wurde, welche Schwierigkeiten im Wege standen und was in der kommenden Woche angestrebt wird. In Abbildung 11 ist das vorläufige Ergebnis zu sehen. Von etwa 90 Studierenden, die wir als aktiv studierend betrachten können, schreiben im Schnitt mehr als 70 pünktlich ihren Blogeintrag.

Diese Blogeinträge sind für uns ein wertvolles Evaluationsinstrument, bei dem wir wöchentliches Feedback darüber erhalten, was in den verschiedenen Fächern bei den Studierenden ”hängengeblieben” ist, wo zusätzlicher Erklärungsbedarf besteht oder ob die

Abbildung 11: Blog-Einträge pro Semesterwoche



Arbeitsbelastung zu hoch ist. Ein Wochenblogeintrag wie: ”In Programmieren und SWE haben wir Klassen und Objekte kennengelernt” zeigt zum Beispiel auch, wie gut die Verzahnung der Module funktioniert. Durch die Einträge wird den Studierenden selbst klar, was sie in der vergangenen Woche gelernt haben:

> *"Diese Woche haben wir uns in STEP mit der Erstellung eines Einheitskreises beschäftigt. Mit Hilfe des Programms ffmpeg, haben wir außerdem mehrere Bilder, mit verschiedenen Varianten des Kreises, zu einem Video zusammenfügen können.*
>
> *Die Wochenaufgabe in STEP empfand ich diese Woche wesentlich leichter, als in der vergangenen Woche und konnte sie auch schnell umsetzen. In Programmieren haben wir das Thema Rekursion wiederholt, hinzu kam das Thema Objektorientierte Programmierung in Java, sowie die Verknüpfung von mehreren Klassen.*
>
> *Weiterhin schwer fällt mir Mathe, aber da muss man halt durch. Zum Glück hilft ja bekanntlich das Internet bei Verständnisproblemen. Ich freue mich jedenfalls auf den Unternehmensbesuch in der nächsten Woche."*

Als ein erfreuliches Ergebnis der Studieneingangsphase kann zudem festgehalten werden, dass die Workshop-ähnliche Lernumgebung über das ganze Semester hinweg eine Wissensvermittlung in einer sehr intensiven Lernatmosphäre ermöglicht hat.

Literatur

[Dennert-Möller u. Garmann 2016] Dennert-Möller, Elisabeth ; Garmann, Robert: Das „Startprojekt“ - Entwicklung überfachlicher Kompetenzen von Anfang an. In: Schwill, Andreas (Hrsg.) ; Lucke, Ulrike (Hrsg.): *HDI 2016 : Hochschuldidaktik der Informatik*, 2016, 11 – 24

[DigiPEER] DigiPEER: *Digitalisierung großformatiger Pläne und technischer Zeichnungen zur Erfassung und Erschließung des Raums*. – http://www.digipeer.de/index.php Zugriff: 02.12.2018

[Key u. Hill 2018] Key, Olivia ; Hill, Lukasz: Die Studieneingangsphase im Umbruch. Anregungen aus den Hochschulen. In: *nexus impulse für die Praxis* (2018), Nr. 14

[Knuth 1998] Knuth, Donald E.: *The Art of Computer Programming*. Bd. 2, Seminumerical Algorithms. 3. Addison-Wesley Professional (Pearson), 1998

[Merkel 2014] Merkel, Dirk: Docker: Lightweight Linux Containers for Consistent Development and Deployment. In: *Linux J.* 2014 (2014), März, Nr. 239. – ISSN 1075–3583

[Papert 1980] Papert, Seymour: *Mindstorms: children, computers, and powerful ideas*. New York, NY, USA : Basic Books, Inc., 1980. – ISBN 0–465–04627–4

[Rivest u. a. 1977] Rivest, R. L. ; Shamir, A. ; Adleman, L.: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. 1977. – http://people.csail.mit.edu/rivest/Rsapaper.pdf

[Robins 2010] Robins, Anthony V.: Learning edge momentum: a new account of outcomes in CS1. In: *Computer Science Education* 20 (2010), Nr. 1, S. 37–71

[Schmolitzky 2017] Schmolitzky, Axel: Zahlen, Beobachtungen und Fragen zur Programmierlehre. In: Brügge, Bernd (Hrsg.) ; Krusche, Stephan (Hrsg.): *Tagungsband des 15. Workshops Software Engineering im Unterricht der Hochschulen*, 2017, 83–90

[Stachowiak 1973] Stachowiak, Herbert: *Allgemeine Modelltheorie*. Bd. 2, Seminumerical Algorithms. Wien : Addison-Wesley Professional (Pearson), 1973. – ISBN 3–211–81106–0

[Steven J. Vaughan-Nichols] Steven J. Vaughan-Nichols: *Linux now dominates Azure*. – https://www.zdnet.com/article/linux-now-dominates-azure/ Zugriff: 30.10.2018

[Vosseberg 2015] Vosseberg, Karin: Mit Projekten ins Studium starten. In: Schmolitzky, Axel (Hrsg.) ; Hauptmann, Anna S. (Hrsg.): *Tagungsband des 14. Workshops Software Engineering im Unterricht der Hochschulen*, 2015, 123–124

[Vosseberg u. a. 2015] Vosseberg, Karin ; Czernik, Sofie ; Erb, Ulrike ; Vielhaber, Michael: Projektorientierte Studieneingangsphase. Das Berufsbild der Informatik und Wirtschaftsinformatik schärfen. In: Schubert, Sigrid (Hrsg.) ; Schwill, Andreas (Hrsg.): *HDI 2014 : Gestalten von Übergängen*, 2015, 169 – 177

[WSV 2018] WSV: *Maritime Verkehrstechnik*. 2018. – https://www.gdws.wsv.bund.de/DE/schifffahrt/03_verkehrstechnik/verkehrstechnik-node.html

Using Mini-Projects to Teach Empirical Software Engineering

Michael Felderer[1] and Marco Kuhrmann[2]

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

Abstract

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

1 Introduction

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

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

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

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

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

2 Background and Related Work

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

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

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

3 Course-Integrated Mini-Projects

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

3.1 Mini-Projects and Project Teams

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

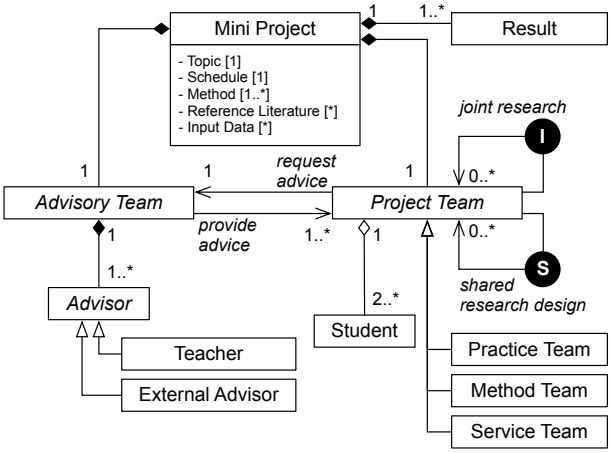


Figure 1: Overall organization model of mini-projects.

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

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

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

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

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

3.2 Project- and Task Descriptions

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

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

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

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

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

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

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

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

Work Description and Work Plan

1.1 Work Description

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

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

Your team will receive the following input material:

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

Your team is expected to deliver the following items:

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

1.2 Basic Schedule and Related Projects

Your project follows the following basic schedule:

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

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

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

Project 12, 13, 16 (further survey projects)

1.3 Further Literature

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

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

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

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

3.3 Course-Integration Strategies

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

3.3.1 Implementation as a Self-Contained Empirical Software Engineering Course

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

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

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

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

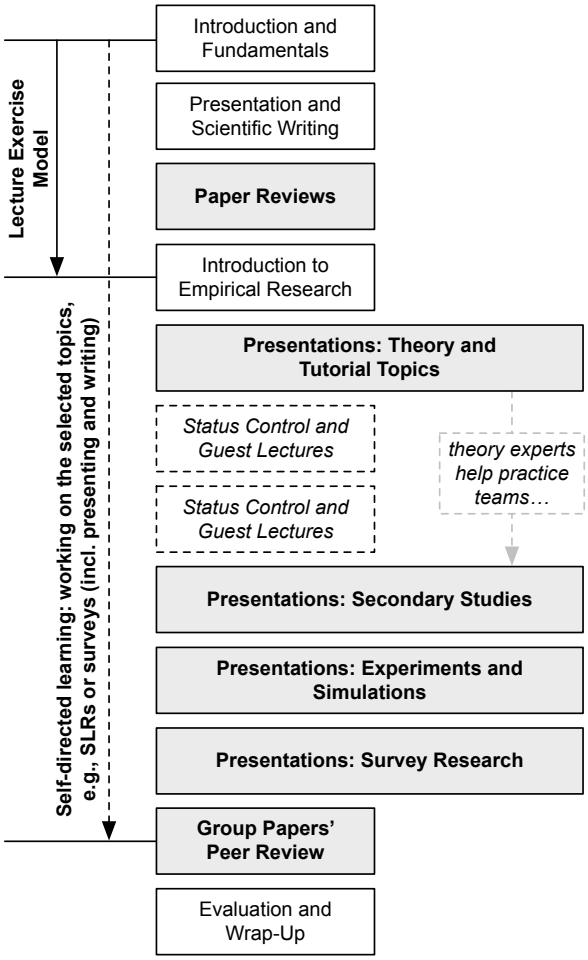


Figure 3: Overview of the topics and the general organization of the SCM course.

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

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

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

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

3.3.2 Integration in an Advanced Software Engineering Course

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

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

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

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

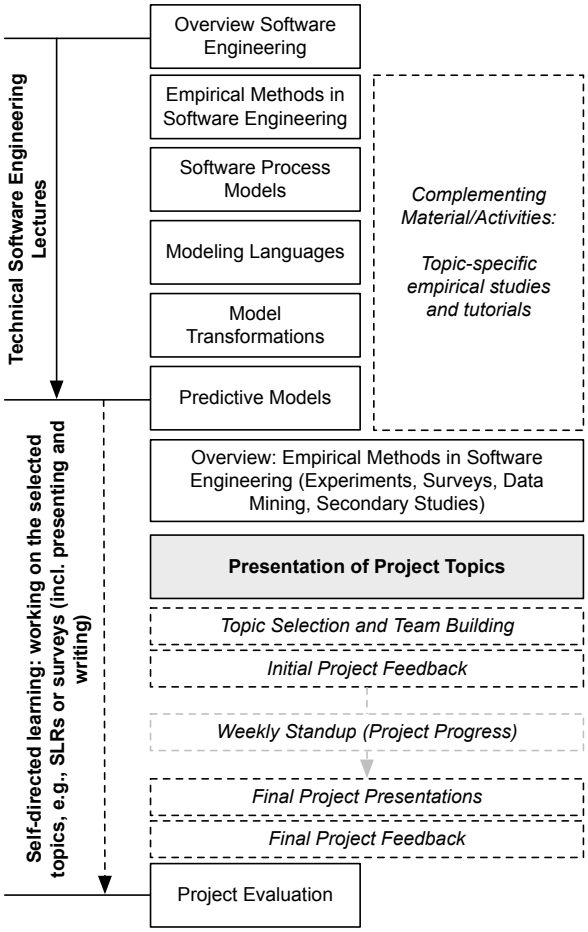


Figure 4: Overview of the topics and the general organization of the ASE course.

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

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

4 Evaluation

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

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

Table 4: Overview of the research questions studied.

4.1 Research Design and Evaluation Strategy

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

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

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

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

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

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

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

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

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

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

Table 5: Results of the final course evaluation.

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

4.2 Results and Discussion

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

4.2.1 RQ 1: Improved Work Approach

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

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

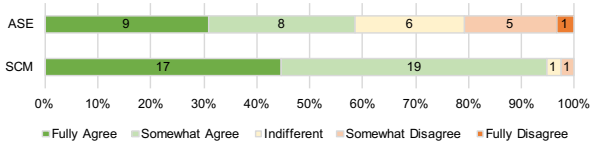


Figure 5: Results for the learning experience from final questionnaires (SCM: n=38, ASE: n=29).

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

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

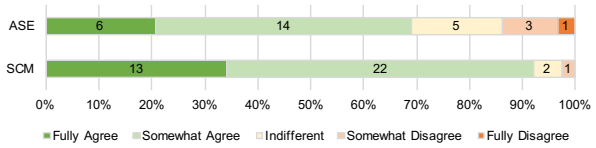


Figure 6: Results for the improved understanding from final questionnaires (SCM: n=38, ASE: n=29).

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

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

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

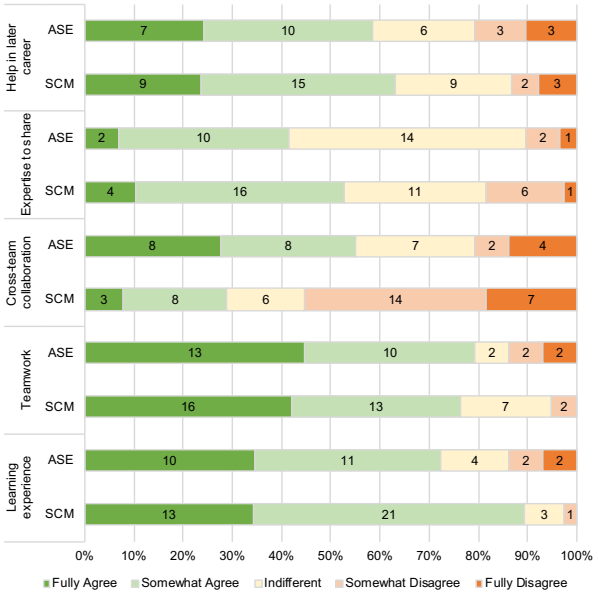


Figure 7: Results for the perceived leanings from final questionnaires (SCM: n=38, ASE: n=29).

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

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

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

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

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

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

4.2.2 RQ 2: Improved Understanding

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

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

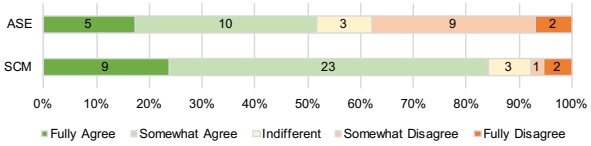


Figure 8: Results for the changed view on science from final questionnaires (SCM: n=38, ASE: n=29).

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

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

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

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

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

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

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

4.2.3 RQ 3: Perceived Dis-/Advantages

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

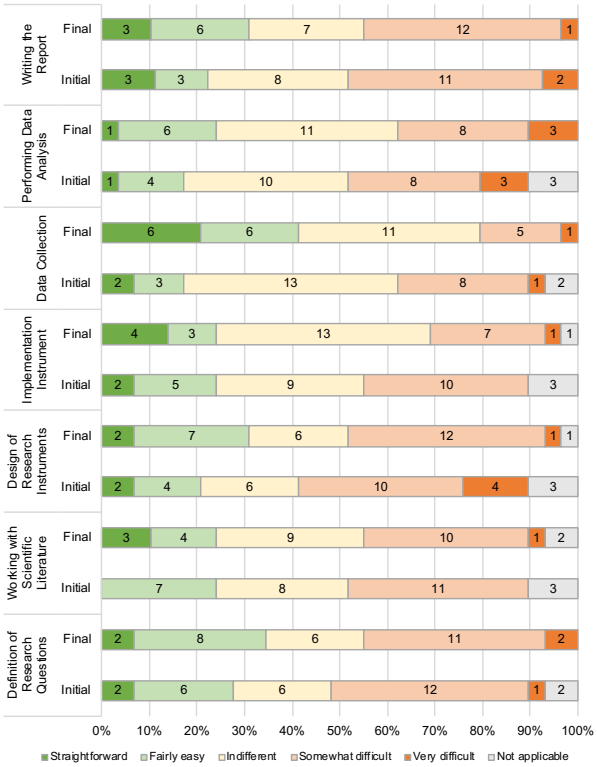


Figure 9: Overview of the perceived challenges on the different scientific work activities in the ASE initial and final questionnaire (n=29).

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

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

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

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

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

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

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

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

4.3 Threats to Validity

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

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

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

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

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

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

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

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

5 Conclusion

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

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] M. Kuhrmann and J. Münch. Distributed software development with one hand tied behind the back: A course unit to experience the role of communication in gsd. In *1st Workshop on Global Software Engineering Education (in conjunction with ICGSE'2016)*. IEEE, 2016.

[18] M. Kuhrmann and J. Münch. When teams go crazy: An environment to experience group dynamics in software project management courses. In *International Conference on Software Engineering*, ICSE, pages 412–421. ACM, May 2016.

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

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

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

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

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

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

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

[26] C. Wohlin. Empirical software engineering: Teaching methods and conducting studies. In *Proceedings of the International Workshop on Empirical Software Engineering Issues: Critical Assessment and Future Directions*, volume 4336 of *LNCS*, pages 135–142. Springer, 2007.

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

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

Experience Report on an Inquiry-Based Course on Model Checking

Sebastian Krings
Universität Düsseldorf, Hochschule Niederrhein
krings@cs.uni-duesseldorf.de
Philipp Körner, Joshua Schmidt
Universität Düsseldorf
{koerner,schmidt}@cs.uni-duesseldorf.de

Abstract

The development and improvement of model checkers for the validation of hard- and software is an ongoing research topic in computer science. Model checking research connects theoretical and practical aspects; new algorithms are often implemented inside well-known model checkers which have been in development for many years. This is seldom taken into account by university courses on the topic, which often remain on the theoretical level. Hence, they do not provide practical access to the topic for reasons such as lack of time or inaccessibility of tools and their source code.

In this article, we present a recent revision of our course on model checking, shifting from a classical lecture-based format to inquiry and research-based teaching. We document course development, present some didactic methods used and evaluate the course based on peer review and student feedback. Furthermore, we try to assert student engagement empirically.

Introduction and Motivation

The development and improvement of model checkers (Clarke u. a., 1999) for the validation of hard- and software is an ongoing research topic in computer science (Grumberg u. Veith, 2008). Model checking research connects theoretical and practical aspects; new algorithms are often implemented inside well-known model checkers which have been in development for many years. This is seldom taken into account by university courses on the topic, which often remain on the theoretical level, not providing practical access for various reasons.

In particular, both complexity and code volume of commonly used model checkers prevent students from coming to grips with internal workings. In consequence, typical courses on model checking stay on a theoretical level. Students often have no way of experiencing the practical aspects of developing a model checker. This leads to shortcomings in different areas:

- Learning results might be reduced due to missing hands-on experience.
- Regarding bachelor's and master's theses the scope of topics is limited, as students have not learned how to appropriately address practical problems in model checker development.
- Missing experience in project work, tool usage and working collaboratively have been identified as major areas in which students do not meet expectations from industry (Radermacher u. a., 2014; Radermacher u. Walia, 2013; Tafliovich u. a., 2015). Those skills could be acquired en passant in a programming project.

To improve, we decided to remodel our course on model checking. The overall goal was to move from classic lectures to active learning techniques for an improved hands on experience. Furthermore, we noticed that students writing theses at our chair sometimes lack the required knowledge about how research is performed and thus need close supervision and initial training. In order to motivate individual research and to enable students to train their skills, we decided to move from a classic lecture setting to inquiry-based learning. In particular, we intended for the course to follow the typical pattern of asking appropriate questions for research, finding evidence or creating it through experiments, compare and discuss results as well as explain and publish differences discovered.

During the course, participants should:

- Acquire the theoretical foundations of model checking by identifying and analyzing common software errors.
- Align these foundations with the body of knowledge.
- Design and implement a novel model checker as independently as possible.

Barron et al. (Barron u. a., 1998) identified four important aspects contributing to the success of inquiry-based courses:

- Selecting appropriate learning goals,
- Begin with problem-based learning before gradually switching to project work,
- Enable self-assessment and revision,
- Develop an atmosphere and social structures that support participation.

Following, we discuss how we realized these aspects in our course.

Increasing practical relevance and self-responsibility aside, the course redesign serves another purpose. By implementing a novel model checker together with the students we rely less on locally developed tools and avoid tailoring towards in-house techniques. Instead, a course like the one outlined can help evaluating alternative technologies together with our students.

Additionally, the course provides other more long term benefits to the chair. First, we can use the resulting software in bachelor's and master's theses, given that it is more easily accessible than ProB (Leuschel u. Butler, 2008, 2003), the model checker developed at our chair at the University of Düsseldorf. In particular, it is written in Java, which most students know from their undergraduate studies, whereas ProB is written in Prolog, a language not as common. Second, it can serve as a playground for trying out more experimental algorithms or backends.

Following, we start with introducing setting and context of the course. Afterwards, we discuss goals and challenges of remodelling the course using an inquiry-based format. In particular, we document preparation and execution phases. An evaluation of student motivation and engagement will be performed as well. Evaluation and our conclusions lead to another iteration of the course which we describe later on, followed by overall conclusions about both iterations.

Course Setting and Context

Our lecture on model checking is aimed at master students of computer science. In particular, the lecture is part of a major in software engineering and programming languages. In consequence, it should enable participants to immediately continue with a masters thesis in model checking.

There is a corresponding lecture on *Safety Critical Systems*, focussing on how to write software specifications and *use* model checkers and provers rather than implement them. The two lectures are often attended one after the other. However, since we do not enforce a particular order, *Safety Critical Systems* can not be considered a precondition for the course documented in this paper. For both courses, attendance is not compulsory.

Learning Objectives

The learning objectives were already predefined and aligned with the overall curriculum. In consequence, we did not intend to change the high-level goals of the course. After attending, students should be able to

- present and compare different techniques for program verification,
- be able to summarize selected scientific literature on program verification and be able to criticize where appropriate,
- write their own specifications and evaluate them,
- decide on appropriate formalisms, algorithms and tools for given verification tasks.

We believe these goals match the requirements stated by Barron et. al. (Barron u. a., 1998).

Former Course

The former course on model checking before the year 2017 was held in a classical lecture-based format. The lectures dealt with the theoretical foundations of model checking. Lectures were accompanied by weekly exercises used to practice. There were no mandatory submissions but of course students were encouraged to work on the exercises independently.

The course heavily relied on theoretical aspects of model checking, i.e., several theorems and lemmas were proven in the lectures while others had to be proven by the students in the exercises.

Students were not supposed to implement any algorithms throughout the course. Instead, the most important algorithms were discussed in pseudo-code within the lecture. Moreover, the students were occasionally asked to create pseudo-code algorithms for simple tasks on their own such as an explicit-state model checking algorithm for invariant checking.

Students were not introduced in detail to any formalism for writing specifications but worked on an abstract level of formal modeling. However, the students have seen some small models in the lecture without paying any attention to the used formalism but rather focussing on the overall structure. Apart from this, the course remained theoretical neither offering any hands-on experience nor addressing issues

that might arise when implementing model checking algorithms.

In retrospect, we think that the majority of the students suffered from this lecture-based format, as it was not motivating them to work on more advanced topics of model checking. We think that focussing on the theoretical aspects of model checking without any hands-on experience might be daunting for students to work on that topic at all and hinders them to use model checking techniques in future projects. These insights have lead us to remodel the course in the next semester, introducing more interactive elements and practical experiences as we will outline below.

Remodeled Course

In the following sections, we will describe the remodeled course in detail, starting with the participants, the goals and milestones of the redesign and the steps performed in preparation, execution and postprocessing. Additionally, we will discuss two teaching methods employed during the course.

Participants

The first iteration of our remodeled course took place in the summer term of 2017. There were 11 students attending, with 6 of them attending regularly. Three students were undergraduates pulling up courses.

Goals

First, students should gain more practical access to model checking and how it connects to software engineering. Furthermore, the course should be connected to recent research results.

In addition to teaching model checking, we want to encourage independent research. In particular, this includes supporting students throughout the common research phases from finding a suitable hypothesis to publication.

Target Milestones

The target milestones are divided in three stages:

1. Planing and preparation.
2. Implementation and supervision of the research project.
3. Publication and presentation of results.

Individual milestones are given in Table 1. The third phase is not completed as of yet: While we ensure reproducibility for upcoming terms, research results have not been published. This is mostly due to dates and deadlines of appropriate software engineering conferences. As a successful publication cannot be guaranteed, it is not an essential part of our lecture and will not be considered further.

So far, three of the six regularly participating students are willing to contribute to a research paper. We will use this opportunity to explain the rules of good scientific practice as well as the common parts of the scientific publication process, e. g., peer-reviewing.

Challenges

Switching from a classic lecture to inquiry-based learning result in a number of challenges:

- Cognitive requirements are higher, as we switch from knowledge reproduction to production.
- Course progression is less linear. In particular, project progress will be fluctuating and has to be monitored closely to allow for timely interventions in case goals might become unreachable otherwise.
- Adding an additional programming task to the curriculum increases the individual workload students are exposed to. In consequence, individual motivation and commitment has to be increased.
- Research has to be controlled in order to avoid getting off track. Students should have as much freedom as possible, while still being guaranteed to reach the intended learning outcomes.
- Due to the higher amount of group work and cooperation, exams have to be prepared carefully to meet both the didactic requirements and the ones posed by exam regulations.

Documentation Preparation

As stated in Table 1, we began course preparation by collecting and inspecting existing lecture material such as slides and exercise sheets. Furthermore, in order to gain additional material, the latest literature on model checking was sighted as well.

Reducing existing material to make room for the programming tasks proved difficult for two reasons:

- It was hard to anticipate the speed with which students would progress. As this is our first inquiry-based attempt at teaching model checking, we could not rely on previous experiences.
- As stated above, we intended for the module to be as open as possible. In particular, we wanted to avoid predetermining techniques and algorithms to be used. However, this made it impossible to anticipate what the students would come up with first. We needed to prepare for several possible pathways through the course.

To reduce overhead, we decided to reduce content even more, until we reach the bare minimum of model checking knowledge. Additionally, we prepared several collections of articles, slides and exercises for different subtopics. As soon as speed and direction of student research can be asserted, we could add or remove collections as needed.

Table 1: Phases, Milestones and Time for Completion

	Preparation	
1.	Inspection of existing material	5 h
2.	Literature research and selection, collect relevant papers and documentation to be supplied to students	5 h
3.	Didactic reduction: make room for programming tasks by reducing course content without omitting learning outcomes	5 h
4.	Prototypical implementation to assert difficulty level and needed effort, identify obstacles	10 h
	Execution	
5.	Project and time management, task distribution, team building (done in lectures)	13 h
6.	Discussion, monitoring of results (outside of lectures)	15 h
7.	Further implementation work, bugfixing (outside of lectures)	13 h
8.	Peer review by didactics department	6 h
	Postprocessing	
9.	Collecting results, documentation	10 h
10.	Benchmarks and performance evaluation	10 h
11.	Publication process	10 h
12.	Evaluation and preparation for upcoming terms	5 h
	Total	107 h

To assert the overall workload and to identify possible obstacles, we developed a prototypical implementation of a model checker suitable for the course. By doing so, we learned that developing a parser for B, i.e. a software that turns the human readable representation of software into a machine readable form, is a more time-consuming task than we initially suspected. Furthermore, we all had attended lectures on *compiler construction* and thus had at least theoretical knowledge in writing parsers. However, as stated above, we could not assume the same of the participating students.

In consequence, we decided to make our prototypical parser available for the students instead of making it a part of the programming project. This allowed us to focus more on model checking itself, rather than spending time on infrastructure. While this somewhat reduces possible learning outcomes, we still feel our decision is justified. This is stressed by the fact that several other lectures, e. g., *compiler construction*, *dynamic programming languages* or *logic programming*, feature the development of parsers.

Documentation Execution

In the following section, we will document our experience in course execution. In particular, we will discuss how we introduced the topic and helped students formulate initial research questions. Following, two key methods used throughout the course are discussed. We conclude with our approach to grading.

Developing a Research Hypothesis

Following the work by Barron et. al. (Barron u. a., 1998), we decided to start our course problem-oriented instead of immediately confronting students with the need to develop a research question themselves[1]. To do so, we followed a four-step approach to introduce model checking:

- Error- and hazard analysis of an elevator control software,
- Introduce specification languages, by discussing a reduced version of the B language (Abrial, 1996).
- Collaboratively develop key definitions such as state spaces and transitions,
- Collaboratively develop a simple explicit-state model checking algorithm.

We decided to rely on B as an input language, mainly because it is the specification language most commonly used at our chair. However, it is a rather complex language with many features students had to learn. In retrospect, a simpler language could have been used without reducing learning outcomes. In consequence, the course was modified for later iterations, as we will discuss later on.

In the following sessions, we switched to a more inquiry-based approach: For instance, the reduced input language did not include a way to specify certain system properties. As students were writing their own software models in order to gain test cases, those shortcomings became obvious. In consequence, students had to come up with language extensions and develop new model checking algorithms to verify them.

[1] For a comparison on inquiry-based and problem-based learning see, for instance, (Oguz-Unver u. Arabacioglu, 2014).

Of course there are different extensions to simple explicit-state model checking. We intended for the students to find them independently as well as share and discuss their findings. To avoid errors, we had four lectures allocated to scientific foundations. In summary, those were barely needed, as our students were able to independently discover and provide foundations. Without further guidance, they managed to do literature collection and research and were able to draw appropriate conclusions for further development. Furthermore, they managed to bridge the gap to their undergraduate studies where needed autonomously.

Teaching Methods

In the following sections, we will discuss two teaching and organization methods, which we used throughout all lectures. The two methods complement each other: The hazard collection helps keeping in mind the big picture. In contrast, the Kanban board is used to structure the programming project into parts and helps to discover and tackle todos in an organized manner.

Collection of Hazards

To motivate the how and why of model checking, we began the first lecture with a simplified hazard analysis. Working in groups, students were supposed to describe behavior scenarios of an elevator that performs as bad as possible. Scenarios were collected and sorted, first by grouping corresponding ideas such as “the elevator never opens its doors” and “the elevator never closes its doors”. The resulting collection is shown in Figure 1a.

Not all of the identified scenarios can be prevented using a model checker. We thus supposed to reorganize our collection by two axes: The dangerousness of the scenario and the extent to which the elevator control software is involved. The sorted collection is shown in Figure 1b. In the upper right corner, we see those scenarios that should be considered first, i. e., those that are hazardous and primarily caused by software defects.

We keep the collection around during the course of the semester, removing those cards representing errors our model checker was able to detect. In consequence, the hazard collection was driving the inquiry-based learning process: While some errors were easy to detect using simple explicit-state model checking techniques, others made more involved techniques necessary. For instance, a property such as “If the elevator is moving, doors are closed” is comparably simple to verify. In contrast, “At some point in time the elevator is moving”, is more involved due to time-based reasoning.

Project Management using Simplified Kanban Boards

To manage programming tasks and how they are distributed between the participants we used a method based on simplified Kanban boards. Every occurring task is collected on a (virtual) flashcard. While being processed, tasks follow a predefined live cycle by moving from and to different stacks:

1. Backlog - Newly created tasks start here. If a task is not actionable, because a precondition is missing, a new flashcard for the precondition is added. If a task is too large or to unwieldy to handle, it is split into multiple subtasks. Tasks that can be approached now are moved to the next stack.

2. To Do / Ready - Includes actionable tasks that can be addressed right now. Students can pick a task and assign it to themselves. Once this happens, the task moves forward.

3. In Development - Tasks in process by someone. Each task in this stack has someone responsible for its handling assigned.

4. Testing / In Review - Task that the assigned students consider finished are moved here. For quality control, another student has to verify whether the task has been handled satisfactorily. This is usually done by adding automatic test cases and by performing manual code review. If problems are identified, the task is moved back to “In Development”. Added test cases and comments should now help find a better solution. Otherwise, the task is moved to the next stack.

5. Done - Tasks that have been finished and verified.

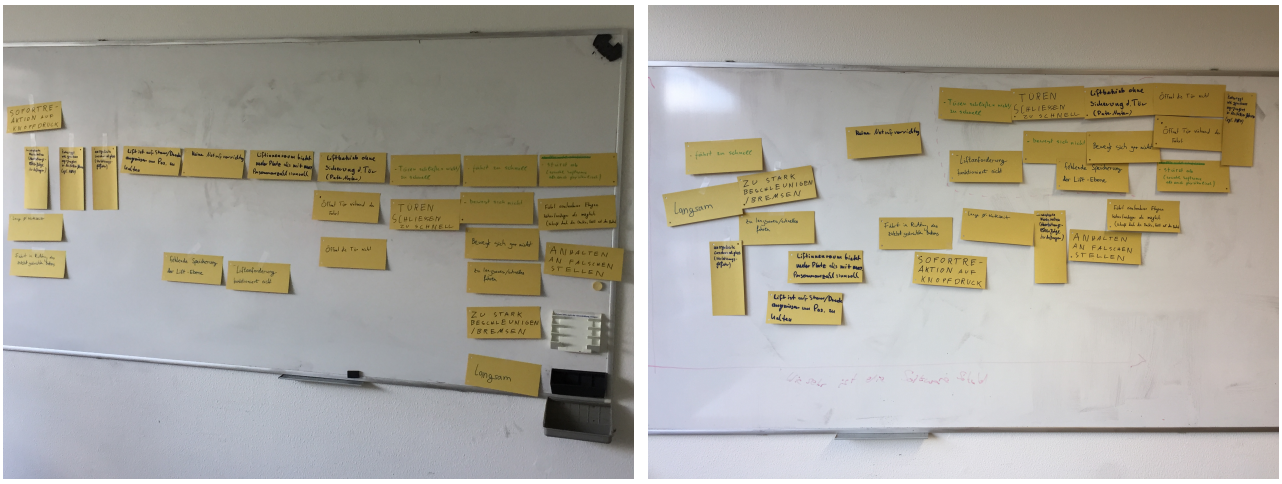
We used a virtual board on GitHub as shown in Figures 2 and 3. The method proved easy to learn for students and efficient in handling programming process. Furthermore, it served as a documentation of participation that could be considered for grading, as we will discuss in the following section.

Grading

As the course was teaching both theoretical and practical aspects of model checker development, exams were supposed to measure both parts. In additional, we had to ensure that grading complies with the examination regulations. In particular, regulations enforce grades to be based on individual performance, i.e., group work can not easily be taken into account.

To improve constructive alignment (Biggs, 1996), we implemented a combination of formative and summative exams:

- Constant participation in the programming project was documented using the Kanban project management method as outlined above. As changes in tasks and issues were recorded together with the student’s username, following the individual contribution was easy. This formative part mostly considered the individual contributions to the programming project and takes into account that different parts of the model checker were developed



(a) By Topic (b) By Hazardousness / Software Influence

Figure 1: Elevator System - Hazard Analysis

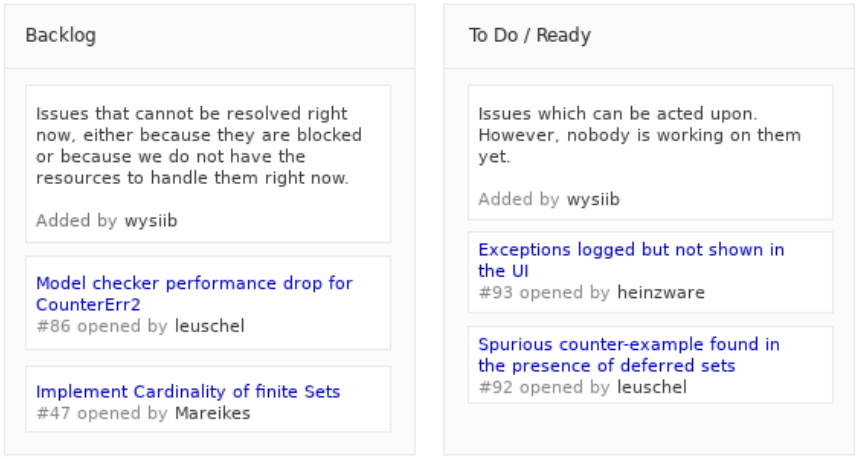


Figure 2: Kanban Board, Part 1

independently. For example, UI and model checking core were developed by different students and at different times.

- Changes in attitude, development of soft skills such as ability to cooperate and stance towards ethical issues in software development could be observed both in the live sessions and the online discussions. Of course, those cannot simply be graded on observation alone and thus mostly served for the lecturers to assert progress.

- Theoretical foundations of model checking were verified by a written summative exam at the end of the semester. Keep in mind that we had to reduce theoretical content to be able to fit in the programming project.

The combined exam is able to verify, whether the learning objectives stated above have been reached:

- The goal to enable the students to present and compare different techniques for program verification was verified using project contributions. Students had to decide between different algorithms (i.e. compare) and they had to present the implementations during the reflection & evaluation sections.

- The goal to enable the students to summarize selected scientific literature on program verification and be able to criticize where appropriate, was partially covered by the reflection & evaluation sections. Additionally, we used the exam to ask students to select an appropriate algorithm for given problems, i.e. to criticize weaknesses ruling out algorithms.

- Students wrote their own specifications and evaluated them in order to create test cases for the programming project.

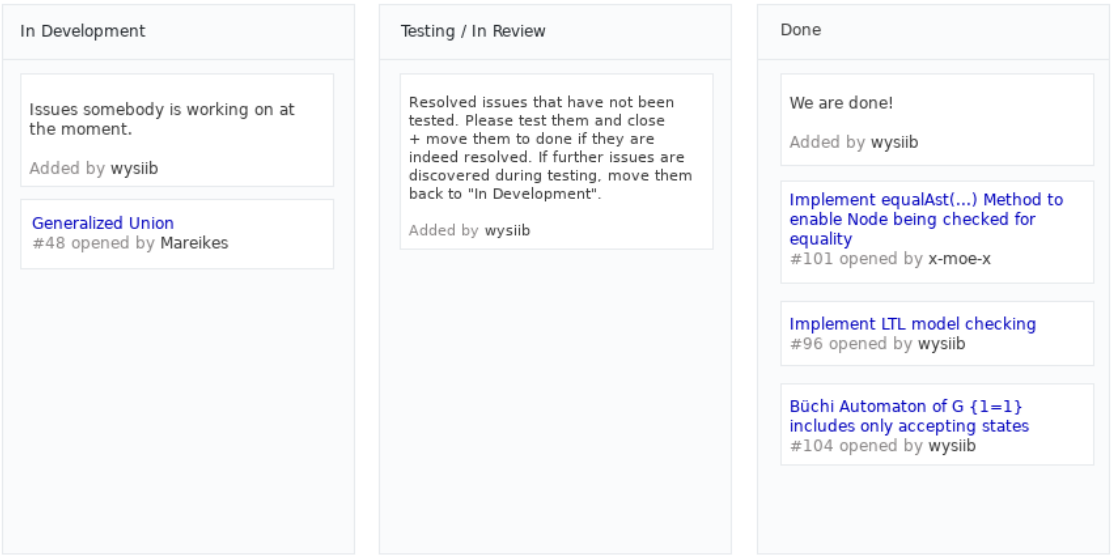


Figure 3: Kanban Board, Part 2

- Again, we used the written exam to ask students to decide on appropriate formalisms, algorithms and tools for given verification tasks.

Course Evaluation

This section will discuss several means of evaluation we employed to measure the success of the redesign. We will discuss review by other faculty members as well as student feedback and engagement.

Peer Review

Two sessions were monitored and reviewed by other faculty members and members of the didactics department. Each time, certain weaknesses were spotted and later fixed based on suggestions of the observers.

R & D Session

The intention of the first observed session was to develop novel model checking techniques in order to be able to detect more of the hazards collected in Figure 1b. In particular, we tried to encourage research and experimentation and avoided discussing known solutions beforehand. We decided to have the session supervised in order to verify if we hit the sweet spot between student research and lecture progress.

The overall concept of the sessions and the belonging teaching methods were evaluated as coherent and aligned with the goals of the sessions. However, weaknesses were spotted when it comes to student interaction and result summarization and recapitulation:

- Sometimes, we failed to ensure that all students had understood the currently discussed idea well enough to participate in further discussions. In consequence, some ideas for new algorithms were developed within smaller groups of students and only later discussed with the group as a whole. For future sessions we decided to discuss and document new ideas more thoroughly, including visualizing them on a flip chart. Resulting documentation should also help later implementation.

- For some ideas the group decided to be worthwhile of further experimentation, the next steps to take remained unclear. In particular, resulting tasks have to be identified immediately and clear commitment to resolving them has to follow.

Reflection & Evaluation Session

The second session that was observed was meant as a synchronization point between different groups working individually. While certain students were working on verification techniques for infinite software models, others were working on time-based reasoning. As discussed, both topics are considered essential for the course. We thus decided to form focus groups with the intent of later updating the other groups.

Furthermore, presenting intermediate results to other groups was intended to account for the appropriate reflection upon the executed research tasks. As pointed out in (Blumenfeld u. a., 1991; Schauble u. a., 1995), project-based work could otherwise focus too much on the execution without evaluation results and lessons learned. In consequence, students would have less opportunity for self-assessment and revision, violating one of the principles of successful inquiry-based courses stated by Barron et. al. (Barron u. a., 1998).

During the lecture, presentation of intermediate results was sluggish, students appeared to be only superficially prepared. In retrospect, we identified our imprecise task statement as the most likely reason. Again, we adapted following sessions in order to improve:

- In addition to asking for a presentation of intermediate research results, we supply an outline:
  1. Short presentation of used techniques and how they are supposed to work.
  2. Summary of difficulties and solutions implemented so far.
  3. Open problems, followed by a discussion of possible solutions.
  4. Code examples and actual implementation where sensible.
- To increase student commitment, outline and presentation content has to be discussed with the lecturers beforehand (see milestone 6 in Table 1).

Student Feedback

Due to the small number of participants, no official evaluation was performed by the university. For the same reason, there are no former evaluations we could compare to.

However, during the course, we performed several intermediate evaluations using short online surveys. Feedback was very positive and encouraging. In particular, the course was described as both interesting and challenging at the same time. Furthermore, students evaluated their own learning achievements as high. More sustainable learning was attributed to the increased self-reliance compared to other courses.

The main point of criticism was the volatile speed of progression during the lectures. While this is not uncommon for research-based processes, it turned out to be the major cause of frustration for students. As an immediate remedy, we started to monitor progress more closely and intervened early once students got stuck. For the next iteration of the course, we believe we should reconsider the balance between problem-based and inquiry-based learning.

In order to gain more insight into workload, motivation and learning outcomes, we performed another feedback round when writing this article.

The student's workload was rated with an average of 3.75 on a scale from 1 (= low) to 5 (= high). This is in line with our expactations and does not suggest that the inquiry-based setup increased the overall workload. Learning outcome was rated with an average of 4.375, while overall motivation was rated with an average of 4.625, both again on a scale from 1 (= low) to 5 (= high).

However, given the low number of course participants, we cannot claim any statistical significance.

Grades

In 2017, six students participated in the exam. Without taking into account the project work, all students passed the exam with an average of 1.43 (equivalent to US 3.9 - 3.7, "excellent"). When taking into account the formative part of the exam, i. e., project work and participation, the average improves to 1.28 (equivalent to US 3.9, "excellent").

Table 2: Average Grades

	2014	2015	2016	**2017**	2018
# Students	2	5	7	**6**	5
∅ Grade	1.85	2.58	1.71	**1.28**	1.88

For comparison, the average grades of different course iterations is given in Table 2. The low average in 2015 is due to one student not passing. If only passing students are counted, the average improves to 1.98.

Judging by the average grades, the highly inquiry-based course in 2017 performed better than the lesson-based course in 2016. Furthermore, the next iteration of the course, which includes several improvements we we will discuss below, again exhibits a slightly worse average grade.

Of course, we would like to attribute the improved grades to the practical research experience, that students could gain during the hands-on sessions. However, with only five to seven students participating in the exams, the sample size is much to small to draw reliable conclusions.

Student Engagement

One of the main challenges was to keep students motivated to participate in research and programming. In particular, participation outside of lecture hours had to be ensured. Given that we used Git to record all changes made to the source code, we can use the data collected for statistics on participation and engagement. Apparently, we were able to motivate participants to consistently and autonomously work towards the course goals.

For instance, this can be seen in Figure 4, showing a so called "punchcard". Time of day on the x-axis and days of the week on the y-axis form a grid in which dots of different diameters are drawn. The larger a dot is, the more extensively the source code was changed at that time interval.

Of course by far the biggest amount of changes was made during lectures, exercises and tutorials, wednesday and friday 12 - 2 p.m. While these are the most active phases, we can see student activity throughout all days and times. Particularly pleasant is the constant activity occurring wednesdays at 7 p.m. and fridays at 8 p.m. Apparently, they are caused by students revising lecture material and incorporating it into the programming project.

Another indicator for high motivation is that the project was both fascinating and motivating enough to cause the occasional all-nighter. For instance, we see source code changes thursdays at 3 a.m. and saturdays at 4 a.m. To determine if this is an indication of motivation or high work load, we asked the students

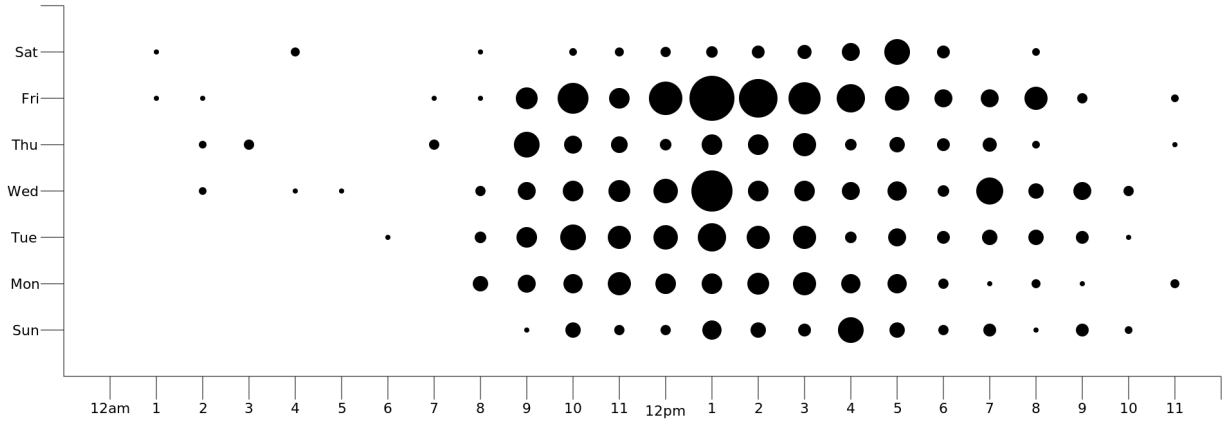


Figure 4: Source Code Changes per Day / Time

to evaluate. Students stated that late night work was not due to high work load or last-minute submissions, but to individual interest, motivation and time management. As we already stated, the workload was rated as 3.75 on average, using a scale from 1 (= low) to 5 (= high).

With the data depicted in Figure 5 we tried to evaluate, whether students were indeed following an inquiry-based approach to the course topics. To do so we tried to evaluate if progress was linear or if students had to follow the typical research pattern of finding approaches, try and experiment and later evaluate. The x-axis shows project progress from February to September. Y-axis shows additions and deletions performed in the source code, i.e., the higher the green part, the more source code has been added to the project. The red part depicts deletions, that is the lower the red bar is, the more source code has been remove from the project.

The small peak at the begin of February is caused by the lecturers developing a prototypical implementation as we discussed for the preparation phase. Lectures itself started in april. In summary, the graph shows that a considerable proportion of the source code added has been removed later on. The amount of lines removed rises and falls with the amount of lines added. This hints at the fact that students did not only add new code once a new idea was developed. Rather, constant reevaluation lead to code being revisited or even replaced. Participants did not progress in a linear fashion.

Another interesting point is the peak mid of June. At that time, students developed certain ideas of "symbolic model checking" and started to implement them in our tool. For this peak, there is no following peak with deletions. This could mean that students made some fundamental progress by reinventing and realizing a classic idea of model checking.

Publication

The last state of inquiry and research, namely publication, was not part of the course for several reasons. First of all, preparing a meaningful publication takes a lot of work and publication cycles can be quite time consuming as well. Thus, it was impossible to fit publishing into the course as well. Second, while publishing teaches a lot about how the scientific community works, it does not contribute to the course topic.

Nevertheless, three of the participating students were interested in writing a follow-up paper and presenting their work to the community. This gave us the opportunity to introduce them to the common publication process, including concepts such as peer review as well as pre- and post-proceedings of conferences.

The overall process went through a number of meetups and discussion sessions:

1. In a first session, we discussed the publication process as it usually takes places.
2. For two sessions, we discussed how to write an interesting paper, mostly following Simon Peyton Jones' advise on the topic (Jones, 2018).
3. We did a brainstorming session in order to identify the key research questions students answered during the course and to assert which of them would be relevant to a larger audience.
4. We had several meetups to discuss the writing progress and to synchronize us.

As the publication process was not part of any official lecture or university event, we were unable to provide credit points to the participants. Yet, all three of them were very enthusiastic about the idea of writing their first article. In retrospect, we believe that it was mostly the fact that we were aiming for a "real scientific workshop", that was highly motivating. Students felt as part of the scientific community and truly as peers among peers.

Figure 5: Source Code Add / Delete by Date

To our joy, the paper (Petrasch u. a., 2018) was accepted for the *18th International Workshop on Automated Verification of Critical Systems*, which took place in Oxford, GB in January 2018. Two students attended the workshop, with one of them giving the presentation. It lead to fruitful discussions both about the topic and about the course itself. As many of those attending were teachers themselves, success reports of our hands-on approach were of particular interest to them.

Lessons Learned - the Next Semester

Two terms after we remodeled the course, the next iteration took place. During the preparation stage, we started reconsidering the course and how to develop it further. In particular, we intended to address the following concerns, some of which occurred in the old course, while some are caused by environmental variables:

- We do not think the presented approach scales to larger groups. A single project for all students seamed infeasible, since 18 students registered and even more showed up in the first session.
- We were concerned that knowledge did not propagate evenly, as students specialized on a subset of topics, e.g. LTL or explicit-state model checking.
- As the B language is a rich formalism, many problems unrelated to model checking might surface.
- Given that theoretical foundations were reduced quite heavily, we feared students working on more advanced topics had to catch up individually.

In consequence, we tried to find a middle ground between focussing on the implementation of a model checker and a lessons-based course.

As knowledge propagation was our most prominent concern, we decided early on that each student should implement a model checker on their own. This is a trade-off between allowing exploration and hands-on experience as well as feature-completeness, as obviously a single student is not expected to do the work that six students did beforehand.

When switching from group projects to individual programming tasks, students do not gain experience in best practices of software development in a team and project management. However, we think that this was more of a side-effect, given that collaborative software development and synchronization in teams is taught in other courses.

Thus, the overall programming workload had to be reduced. To do so, we introduced two changes. First, we changed the formalism from B to petri nets (Petri, 1966), which are considerably easier to understand and implement: As far as the language interpreter is concerned, only addition and subtraction is required. This allows to focus on model checking techniques instead of writing a more complex language interpreter.

However, writing meaningful test cases is harder in such a low-level formalism. Therefore, we used benchmarks from the annual model checking contest (Kordon u. a., 2016) instead of making students writing their own. As in the previous iteration, we provided a parser for the input format.

Furthermore, we split the programming project into two parts. In the first part of the project, the students had to implement a very basic explicit-state model checker to check for deadlock freedom and to verify

certain safety properties. This part of the project had to be passed in order to receive the permission to write the final exam. The second part of the project dealt with more advanced topics: The existing model checker had to be be extended in order to verify linear-time properties by implementing algorithms that have been introduced in the lecture. We graded the second part of the programming project of each student and incorporated the result into the overall grade with a weight of 25 %.

Secondly, we brought back some of the theoretical lessons and exercises, yet cutting down on enormous proofs so there was enough time to spare to let our students work on their model checkers. We intended that the theoretical exercises in particular deepen the knowledge and allow pointing out connections between individual components throughout the lectures.

Thirdly, in addition to the parser, we provided some components of the model checker that were not covered in the lecture and required cumbersome implementation of algorithms that are only sparsely documented in literature.

In the exam, we focused on theoretical problems instead of applying basic algorithms since each student implemented them already. Thus, given that we think the exam was harder overall, we are pleased with the result in 2018 as shown in Table 2.

During supervision and discussing of the model checking projects, it became clear that many students procrastinated and put off work until the last few weeks, not making use of the available time given by cutting back lectures. This might be caused by the lack of fruitful discussion about how components should be implemented that results in inspiration to try it out and peer-pressure to reach common goal during the next week. A possible solution might be to enforce peer-programming during practical sessions.

Conclusion

In summary, we believe that the restructured course on “model checking” met our self-set goals. Realization was more hassle-free than we anticipated.

In particular, contrary to our concerns, we had no problems motivating students to active participation. Even though the course was quite experimental at times, everybody was willing to try. We only had to intervene with speed and progress direction seldomly.

One of the key challenges proved difficult however. Due to the scale of the programming projects, students had to form groups concerned with different partial aspects. For instance, some were working on techniques for the verification of temporal properties while other were working on performance improvements. However, all students were supposed to take the same exam.

As planned, we had regular meetings dedicated to synchronizing the different groups. However, specialist knowledge remains and is not distributed equally throughout the participants. While the exam and its results show that all participants reached the desired learning outcomes, we think that with a better focus on knowledge transfer an even better outcome could have been reached.

To summarize, our refurbished lecture on *model checking* is gradually being refined. It is intended to replace the former lessons-only course in following iterations.

We believe that a few insights can be carried over from our course to other courses. In particular, we believe that a strong connection between teaching and research is highly beneficial and can be achieved with little overhead using programming projects. Depending on the scenario, a switch to inquiry-based learning is feasible and leads to high motivation among participants, without causing too much reduction in course content. However, focus has to be given to the synchronization of participants, i.e., teachers have to ensure that nobody is left behind.

Last, our course shows that with the right motivation and proper support, inquiry-based learning methods can produce research results on a very high level.

Acknowledgments

The authors would like to thank the university didactics department of the Heinrich-Heine-University for their constant support and consulting. In particular, the authors would like to thank Susanne Wilhelm for the project supervision and Jens Bendisposto and Janine Golov for peer reviewing the lectures.

References

[Abrial 1996] ABRIAL, J.-R.: *The B-book: Assigning Programs to Meanings*. New York, NY, USA : Cambridge University Press, 1996

[Barron u. a. 1998] BARRON, Brigid J. ; SCHWARTZ, Daniel L. ; VYE, Nancy J. ; MOORE, Allison ; PETROSINO, Anthony ; ZECH, Linda ; BRANSFORD, John D.: Doing With Understanding: Lessons From Research on Problem- and Project-Based Learning. In: *Journal of the Learning Sciences* 7 (1998), Nr. 3-4, S. 271–311

[Biggs 1996] BIGGS, John: Enhancing Teaching through Constructive Alignment. In: *Higher Education* 32 (1996), Nr. 3, S. 347–364

[Blumenfeld u. a. 1991] BLUMENFELD, Phyllis C. ; SOLOWAY, Elliot ; MARX, Ronald W. ; KRAJCIK, Joseph S. ; GUZDIAL, Mark ; PALINCSAR, Annemarie: Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. In: *Educational Psychologist* 26 (1991), Nr. 3-4, S. 369–398

[Clarke u. a. 1999] CLARKE, Edmund M. Jr. ; GRUMBERG, Orna ; PELED, Doron A.: *Model Checking*. Cambridge, MA, USA : MIT Press, 1999

[Grumberg u. Veith 2008] GRUMBERG, Orna (Hrsg.) ; VEITH, Helmut (Hrsg.): *25 Years of Model Checking: History, Achievements, Perspectives*. Berlin, Heidelberg : Springer-Verlag, 2008

[Jones 2018] JONES, Simon P.: *How to write a great research paper*. https://www.microsoft.com/en-us/research/academic-program/write-great-research-paper/, 2018. – Accessed: 2018-10-24

[Kordon u. a. 2016] KORDON, Fabrice ; GARAVEL, Hubert ; HILLAH, Lom M. ; PAVIOT-ADET, Emmanuel ; JEZEQUEL, Loïg ; RODRÍGUEZ, César ; HULIN-HUBARD, Francis: MCC'2015 – The Fifth Model Checking Contest. (2016), S. 262–273

[Leuschel u. Butler 2003] LEUSCHEL, Michael ; BUTLER, Michael: ProB: A Model Checker for B. In: *Proceedings FME'03*, Springer, 2003 (LNCS 2805), S. 855–874

[Leuschel u. Butler 2008] LEUSCHEL, Michael ; BUTLER, Michael: ProB: an automated analysis toolset for the B method. In: *Int. J. Softw. Tools Technol. Transf.* 10 (2008), Nr. 2, S. 185–203

[Oguz-Unver u. Arabacioglu 2014] OGUZ-UNVER, Ayse ; ARABACIOGLU, Sertac: A comparison of inquiry-based learning (IBL), problem-based learning (PBL) and project-based learning (PJBL) in science education. 2 (2014), 07, S. 120–128

[Petrasch u. a. 2018] PETRASCH, Jessica ; OEPEN, Jan-Hendrik ; KRINGS, Sebastian ; GERICKE, Moritz: Writing a Model Checker in 80 Days: Reusable Libraries and Custom Implementation. In: *ECEASST* (2018)

[Petri 1966] PETRI, Carl A.: *Communication with automata*, Universität Hamburg, Diss., 1966

[Radermacher u. Walia 2013] RADERMACHER, Alex ; WALIA, Gursimran: Gaps Between Industry Expectations and the Abilities of Graduates. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. New York, NY, USA : ACM, 2013 (SIGCSE '13), S. 525–530

[Radermacher u. a. 2014] RADERMACHER, Alex ; WALIA, Gursimran ; KNUDSON, Dean: Investigating the Skill Gap Between Graduating Students and Industry Expectations. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA : ACM, 2014 (ICSE Companion 2014), S. 291–300

[Schauble u. a. 1995] SCHAUBLE, Leona ; GLASER, Robert ; DUSCHL, Richard A. ; SCHULZE, Sharon ; JOHN, Jenny: Students' Understanding of the Objectives and Procedures of Experimentation in the Science Classroom. In: *The Journal of the Learning Sciences* 4 (1995), Nr. 2, S. 131–166

[Tafliovich u. a. 2015] TAFLIOVICH, Anya ; PETERSEN, Andrew ; CAMPBELL, Jennifer: On the Evaluation of Student Team Software Development Projects. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA : ACM, 2015 (SIGCSE '15), S. 494–499

Evaluation des Arbeitsprozess-integrierten Projektstudiums

Frank Fuchs-Kittowski, Juliane Siegeris und Jörn Freiheit, HTW Berlin

{frank.fuchs-kittowski, juliane.siegeris, joern.freiheit}@htw-berlin.de

Zusammenfassung

Für das Arbeitsprozess-integrierte Projektstudium im berufsbegleitenden Masterstudiengang „Professional IT-Business“ der Hochschule für Technik und Wirtschaft Berlin wurde eine Evaluation durchgeführt. Mithilfe einer Umfrage unter den Studierenden sollte die Wirksamkeit des Konzepts des Arbeitsprozess-integrierten Projektstudiums überprüft werden. Es wurden acht Hypothesen aufgestellt. Die Überprüfung dieser Hypothesen sollte dazu dienen, mögliche Fehlannahmen und Brüche im Konzept zu erkennen. An der Umfrage haben 20 der 31 Studierenden teilgenommen, die das Projektstudium bisher absolviert haben. Die statistische Relevanz ist somit begrenzt, jedoch erlaubt die deskriptive Analyse der Umfrageergebnisse Rückschlüsse auf Änderungs- und Verbesserungspotenziale des dem Projektstudium zugrundeliegenden didaktischen Konzepts. Neben der Beschreibung der Methodik und der erzielten Ergebnisse werden Schlussfolgerungen aus der Auswertung gezogen sowie Ausblicke auf zukünftige Anwendungen des Arbeitsprozess-integrierten Projektstudiums gegeben.

Abstract

An evaluation was carried out for the working process-integrated project study in the in-service master programme "Professional IT-Business" of the University of Applied Sciences Berlin (HTW Berlin). A survey conducted amongst students aimed to evaluate the effectiveness of the concept of work process-integrated project study. Eight hypotheses were set up. The review of these hypotheses should serve to detect possible misconceptions and breaks in the concept. 20 of the 31 students who have completed the project study participated in the survey. The statistical relevance is thus limited, but the descriptive analysis of the survey results allows conclusions to be drawn on the potential for change and improvement of the didactic concept underlying the project study. In addition to the description of the methodology and the results obtained, conclusions are drawn from the evaluation as well as prospects for future applications of the work process-integrated project study.

Einführung

Lebenslanges Lernen ist in der heutigen Gesellschaft unerlässlich. Damit Lernen ein Leben lang – und damit an verschiedenen Orten und in unterschiedlichen Institutionen - gelingen kann, ist die Durchlässigkeit zwischen den verschiedenen (Lern- und Arbeits-) Institutionen und Orten von zentraler Bedeutung. Doch Durchlässigkeit - insbesondere zwischen dem beruflichen und dem hochschulischen Bildungsbereich – wird oftmals reduziert auf die Anrechnung beruflich erworbener Kompetenzen (Freitag et al., 2011), insb. beim Übergang vom Beruf in das Studium, d.h. vertikal (Buhr et al., 2008).

Ein großes, bisher aber kaum betrachtetes Potenzial besteht aber in der horizontalen Durchlässigkeit, die dadurch entsteht, dass Studierende oftmals neben- oder auch hauptberuflich arbeiten. Dabei wenden sie in der Hochschule erlerntes Wissen und Fähigkeiten in der Arbeit an. Aber vor allem erwerben sie in der Arbeit auch neues Wissen und Kompetenzen, die sie bisher noch kaum in ihr Hochschulstudium einbringen können. Um dieses Potenzial zu heben, erfordert es neuartiger Konzepte zur stärkeren Verbindung unterschiedlicher Lernorte (insb. Lernort Hochschule mit Lernort Arbeitsplatz in Unternehmen).

Um gezielt die berufliche Handlungskompetenz von Studierenden zu fördern und dabei den Lernort Unternehmen zu integrieren, wurde an der HTW Berlin ein innovatives Konzept für die Durchführung des „Projektstudiums“ im Rahmen des berufsbegleitenden Masterstudiengangs „Professional IT-Business“ entwickelt (Fuchs-Kittowski et al., 2017): Die Studierenden im Projektstudium lernen dabei im Arbeitsprozess, in realen Projekten (Praxisprojekt) in (ihren) Unternehmen. Das Lernen in der Arbeit gestalten die Studierenden dabei selbst, werden aber umfassend personell und technisch unterstützt. Ein prozess-orientiertes Curriculum, sog. Referenzprojekt, gibt die Lernziele vor und strukturiert das Projektstudium. Es dient der Auswahl des

Praxisprojekts, der Planung der Arbeits- und Lernprozesse und schließlich auch dem Nachweis des erfolgreichen Projektstudiums. Alle Prozesse des Referenzprojekts müssen nachweislich beherrscht werden, indem sie bewältigt und dabei reflektiert und dokumentiert werden.

In diesem Beitrag sollen die wesentlichen Ergebnisse der Evaluation dieses Konzepts des Arbeitsprozess-integrierten Projektstudiums dargestellt werden. Das Hauptziel der Evaluation lag auf der Prüfung der Annahmen bzw. Wirksamkeit des Konzepts des Arbeitsprozess-integrierten Projektstudiums und der Erarbeitung von Optimierungsvorschlägen.

Dieser Beitrag ist wie folgt strukturiert: Nach der Vorstellung des Hintergrunds der Untersuchung (Masterstudiengang, Konzept des Projektstudiums, Evaluation) sowie der verwendeten Methodik werden die Ergebnisse der Umfrage präsentiert und diskutiert und mit den Ergebnissen die Gültigkeit der acht vorgestellten Hypothesen evaluiert. Es werden Schlussfolgerungen für die zukünftige Anwendung des Konzeptes der Arbeitsprozess-integrierten Projektstudien gezogen und diskutiert. Eine Zusammenfassung und ein Ausblick schließen den Beitrag.

Hintergrund

In diesem Kapitel wird der Masterstudiengang „Professional IT-Business“ und das Konzept des „Projektstudiums“ kurz vorgestellt sowie das diesem Beitrag zugrundeliegende Verständnis einer Evaluation erörtert.

Masterstudiengang

Der Masterstudiengang „Professional IT-Business“ wird berufsbegleitend durchgeführt. Die Studierenden erwerben insgesamt 90 Leistungspunkte bei einer Regelstudienzeit von 4 Semestern. Davon werden allein 30 Leistungspunkte für die drei Projektstudien vergeben, die in den ersten drei Semestern durchgeführt werden. Neben dem Projektstudium I (IT-Spezialist/-in) werden im ersten Semester Module zu den Themen Cloud Computing, Data Analytics und Requirements Engineering durchgeführt, im zweiten Semester wird das Projektstudium II (Data Analyst/-in) von den Veranstaltungen Mobile Computing und Enterprise Architecture begleitet und im dritten Semester werden neben dem Projektstudium III (Enterprise Architecture Manager/-in) die Veranstaltungen IT-Security und IT-Controlling gelehrt. Im vierten Semester erstellen die Studierenden ihre Masterarbeit. Die Durchführung der Projektstudien sowie die Erstellung der Masterarbeit erfolgt hauptsächlich in den Unternehmen berufsbegleitend von Montag bis Donnerstag. Die anderen Module finden als Präsenzveranstaltungen freitags und samstags an der Hochschule statt. Der erste Jahrgang in diesem Masterstudiengang wurde zum Wintersemester 2016 immatrikuliert.

Konzept des Projektstudiums

Das Konzept für das Projektstudium (Fuchs-Kittowski et al., 2017) fokussiert auf die Integration von Lernen und Arbeiten. Dabei findet das Lernen und Arbeiten nicht in der Hochschule an einer Aufgabe eines Unternehmens statt, sondern integriert in reale Arbeitsprozesse in den Unternehmen.

Zur Systematisierung und Unterstützung solcher Lernprozesse am Arbeitsplatz, im Arbeitsprozess, in realen Projekten wurden a) prozessorientierte Curricula („Referenzprojekt“), b) eine Vorgehensweise für das Lernen im Arbeitsprozess („Praxisprojekt“) sowie c) organisatorische und technische Instrumente zur Unterstützung der im Arbeitsprozess Lernenden entwickelt (Fuchs-Kittowski et al., 2017):

Lernprozesse werden durch Herausforderungen (Wissenslücken) in realen Projekten („Praxisprojekt“) initiiert. Diese Herausforderungen sollen durch die Studierenden weitgehend selbstgesteuert bewältigt werden. Dabei sind bestimmte Arbeitsprozesse zu durchlaufen. Dies sind für ein Tätigkeitsprofil (z.B. Data Analyst/-in) typische Tätigkeiten, die in einem profiltypischen „Referenzprojekt“ beschrieben sind, das die curriculare Grundlage und Struktur der Qualifizierung (Projektstudium) bildet.

Personale Unterstützung erhalten die Studierenden für das selbst gesteuerte Lernen während der Bearbeitung ihrer Praxisprojekte durch verschiedene Rollen, wie Fachberater, Lernprozessbegleiter, Vorgesetzte und Kollegen/Kommilitonen. Eine technische bzw. mediale Unterstützung erhalten die Studierenden in Form einer Lernplattform, die orts- und zeitunabhängiges Lernen und Arbeiten unterstützt sowie eine individualisierte Betreuung und Organisation der Teilnehmer ermöglicht. Zudem nehmen die Studierenden im Semester zuvor an einer klassischen Lehrveranstaltung (Vorlesung mit Übung) zum relevanten Themengebiet (z.B. Data Analytics) teil, in denen das erforderliche Fachwissen vermittelt wird.

Die Prüfung der Studierenden erfolgt nicht, wie an einer Hochschule traditionell üblich, durch eine Prüfung (Klausur etc.), sondern anhand einer von jedem Studierenden einzeln erstellten Dokumentation über die selbständige und kompetente Durchführung von im Referenzprojekt vorgegebenen Arbeitsprozessen.

Im Konzept und der Bewertung des Projektstudiums spielt die individuelle Reflektion des Arbeits- und Lernprozesses eine wichtige Rolle. Donald Schön prägte den Begriff der *reflective practice* (Schön, 1983). Er plädiert für die bewusste und systematische Reflektion der eigenen Erfahrungen und

argumentiert, dass erst durch die Reflektion der Handlungen - während des Prozesses und danach - ein Lernprozess ermöglicht wird. In vielen Publikationen (u.a. Upchurch & Sims-Knight, 1999; Dingsoer, 2005; Prior et al., 2014) wird seitdem diskutiert, wie Reflektion in die Lehre integriert und aktiv vermittelt werden kann. Eine Zusammenfassung verschiedener Ansätze im Bereich Software-Entwicklung bietet (Burden & Steghöfer, 2019).

Im vorgestellten Projektstudium wird Reflektion zu verschiedenen Anlässen ermöglicht und eingefordert. Dazu zählen regelmäßige Reflektionsgespräche, die Zwischenpräsentation, ein Fachgespräch am Ende des Projekts, insbesondere jedoch die abschließende Dokumentation. Mittels Fragen zu den durchgeführten Prozessschritten werden die Studierenden angeleitet, in der Ich-Perspektive über Projekterfahrungen, Entscheidungen und aufgetretene Schlüsselsituationen zu berichten und die eigenen Aktivitäten zum Referenzprozess in Beziehung zu setzen. Anders als bei einem Projekt-Ergebnisbericht soll diese Art der Dokumentation (ähnlich dem *postmortem review*, vgl. Dingsoer, 2005) den Studierenden helfen, sich ihrer Lernerträge bewusst zu werden.

Evaluation

Dieser Beitrag beschäftigt sich mit der Evaluation des Arbeitsprozess-integrierten Projektstudiums, das bislang vom ersten Jahrgang komplett (Projektstudium I-III) und vom zweiten Jahrgang bis einschließlich des Projektstudiums II absolviert wurde. Evaluation im Sinne dieses Beitrages bedeutet: die praxisnahe Beurteilung des Projektstudiums mit empirischen Methoden (Wottawa & Thierau, 1990).

Eine Evaluation dient der rückblickenden Wirkungskontrolle, der vorausschauenden Steuerung und dem Verständnis von Situationen und Prozessen. Darauf aufbauend können untersuchte Prozesse angepasst und optimiert werden (Götz, 1993). Das Hauptziel der Evaluation des Projektstudiums lag daher auf der Prüfung der Annahmen bzw. der Wirksamkeit des Konzepts des Arbeitsprozess-integrierten Projektstudiums. Die zentralen Grundannahmen dieses Konzepts sollten empirisch geprüft werden. Die Überprüfung der theoretischen Annahmen mit der Praxis soll insbesondere dazu dienen, mögliche Fehlannahmen und Brüche im Konzept zu erkennen sowie Hinweise für Änderungen oder Erweiterungen des Konzepts zu erhalten.

Methodik

Dieser Abschnitt beschreibt die Methodik der Untersuchung zur Evaluation des Projektstudiums.

Untersuchung

Die Untersuchung sollte als schriftliche Online-Befragung der Studierenden durchgeführt werden.

Unter der Zielsetzung der empirischen Überprüfung der Grundannahmen des Konzepts des Arbeitsprozess-integrierten Projekt-Studiums wurden zu den o.g. Teilbereichen der Konzeption (a) Rolle der Prozessorientierten Curricula (Referenzprojekte), b) Lernen im Arbeitsprozess (Praxisprojekte) und c) Personale, technische und fachliche Unterstützung des Lernens in der Arbeit) sowie zur allgemeinen Akzeptanz des Projektstudiums (Gesamtbewertung) Hypothesen gebildet. Zu jeder der Hypothesen wurden entsprechende Indikatoren definiert. Auf Basis der Indikatoren wurden dann die Erhebungsinstrumente entwickelt, eingesetzt und ausgewertet. Zudem flossen auch Erfahrungen aus der Durchführung des Projektstudiums in die Interpretation der Ergebnisse ein, vgl. (Brand, 2014; Mattauch & Kubath, 2005).

Datenerhebung

Der entstandene Fragebogen wurde Online (Survey-Monkey) im Oktober 2018 an die Studierenden verteilt, die das Projektstudium im Sommersemester 2018 durchgeführt hatten. Dies waren 16 Studierende aus dem aktuellen 3. Semester des Studiengangs sowie 15 Studierende aus dem vorherigen Jahrgang, also insgesamt 31 Studierende. Die Fragebogenerhebungen erfolgten anonymisiert.

Datenauswertung

Insgesamt wurde der Fragebogen von 20 Studierenden ausgefüllt (Rücklaufquote: 64,5%). Die Daten wurden mit den im Online-Befragungs-Tool (Survey-Monkey) verfügbaren Werkzeugen ausgewertet. Aufgrund der relativ kleinen Teilnehmerzahl wurde auf die sonst üblichen parametrischen Analyseverfahren verzichtet. Eine überwiegend deskriptive Analyse der Daten, insb. die Angabe von Absolutwerten und der prozentualen Verteilung, erschien zweckmäßig.

Teilnehmer (Stichprobe)

Von den 20 Teilnehmern haben 12 im Sommersemester das Profil „Data-Analyst/in“ durchgeführt, befinden sich also aktuell im 3. Semester. Die Studierenden im vorhergehenden Jahrgang hatten im Sommersemester 2018 die Wahl zwischen den Profilen „Software-Developer“ (2), „IT-Project-Coordinator“ (3) und „IT-Solution-Developer“ (3).

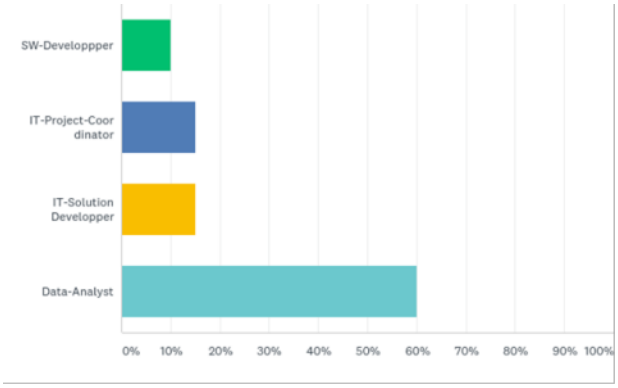


Abb. 1: Gewähltes Referenzprojekt im SoSe2018

Die meisten Teilnehmer waren vor dem Beginn des Master-Studiums an der HTW Berlin bereits 2-3 Jahre in der IT-Branche tätig (47,37%, n=9/19), ein weiterer großer Teil (42,11%, n=8/19) mehr als 4 Jahre und nur ein geringer Teil nahm das Studium nach weniger als 1 Jahr auf (10,53%, n=2/19).

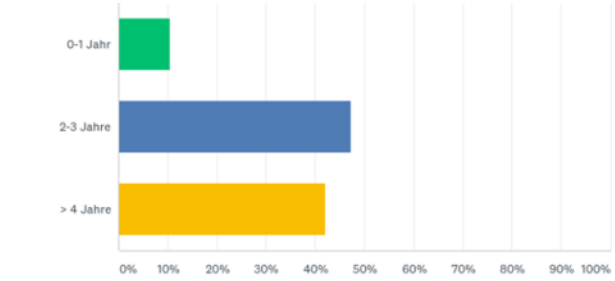


Abb. 2: Berufsjahre der Teilnehmer vor Studium

Aussagekraft

Einschränkend ist anzumerken, dass aufgrund der geringen Stichprobenzahl (n = 20) die statistische Relevanz begrenzt ist. So sind die Aussagen aufgrund der Stichprobe nicht repräsentativ und auf ein Projektstudium mit anderen Rahmenbedingungen (Unternehmen, regionale Bedingungen etc.) nicht bzw. nur bedingt übertragbar. Die erhobenen Daten erlauben aber erste Hinweise auf eine empirische Stabilität des Konzepts.

Unter diesen Voraussetzungen erbrachte die Evaluation die im folgenden Kapitel dargestellten Ergebnisse.

Ergebnisse

Die Grundannahmen des Konzepts wurden in acht Hypothesen zusammengefasst, für die jeweils geeignete Indikatoren definiert worden waren.

Hypothese 1: Referenzprojekte sind eine gute Hilfe zur Identifizierung und Auswahl von geeigneten Praxisprojekten

Als **Hypothese 1** wurde angenommen, dass das Referenzprojekt eine wesentliche Orientierungs- und Unterstützungshilfe zur Identifikation und Auswahl von Praxis-Projekten darstellt. Die Studierenden wurden konkret gefragt, ob sie „anhand der im Referenzprojekt beschriebenen Teilprozesse ein Praxisprojekt identifizieren konnten, das sich für das Projektstudium eignete“.

Fast allen Teilnehmern ist es gelungen, ein geeignetes Praxisprojekt zu finden. Bei 30% der Teilnehmer geschah dies „Völlig unproblematisch“ (n=6) und bei 40% „Relativ leicht und mit geringem Aufwand (n=8). Nur 25% der Teilnehmer hatten „große Mühe und Aufwand (n=5) und nur einem Teilnehmer (5%) ist es „nicht gelungen, ein geeignetes Praxisprojekt zu finden“ (n=1). Bei Letzterem konnte kein Projekt zum Profil „Data-Analyst/in“ gefunden werden, was in der Tat schwierig sein kann, da sich Unternehmen bislang nicht alle mit Data-Analytics beschäftigen.

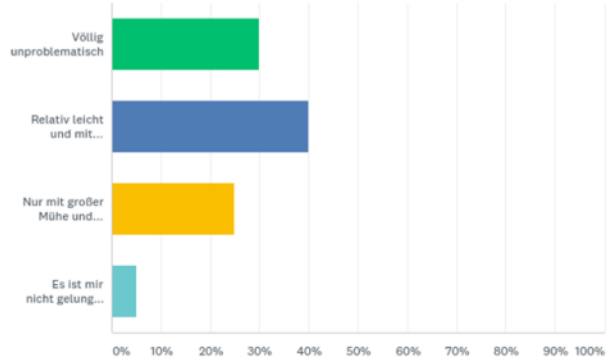


Abb. 3: Einfachheit Praxisprojekt zu finden

Gründe für die aufgetretenen Schwierigkeiten sehen die Studierenden vor allem in:

„keine Daten in ausreichender Menge/Qualität vorhanden, Datenschutz.“

„In der Unternehmensabteilung gehört das angebotene Profil (hier Data Analyst) nicht zum Tagesgeschäft.“

„Es war schwierig die geplante theoretische Herleitung im Projekt der Arbeit umzusetzen, da das Vorgehen des Unternehmens ein anderes ist.“

Zusammengefasst stellten die Referenzprojekte eine wesentliche Unterstützungshilfe zur Identifikation von Praxisprojekten dar, wobei Probleme eher darin bestanden, dass weder im Unternehmen selbst, noch bei derzeitige Kunden Projekte mit dem gesuchten Profil bearbeitet wurden. Zur weiteren Unterstützung der Praxisprojektauswahl erscheint es daher sinnvoll, zukünftig die Anzahl möglicher

Referenzprojekte zu erweitern und die Wahl über alle Semester hinweg zu öffnen.

Hypothese 2: Referenzprojekte dienen als Strukturierungshilfe für das Lernen im Praxisprojekt

Als **Hypothese 2** wurde angenommen, dass das Referenzprojekt eine wichtige Orientierungs- und Strukturierungshilfe für das Lernen in den Praxisprojekten darstellt. Hierzu wurde erhoben, „wie hilfreich das Referenzprojekt bei der Durchführung des Praxisprojekts empfunden wurde“ sowie, ob „geplant ist, das Referenzprojekt auch in Zukunft zur Strukturierung der Arbeit bzw. ähnlicher Projekte einzusetzen“.

Die Mehrzahl der Teilnehmer (40%, n=8) gab an, dass das Referenzprojekt „Überwiegend hilfreich“ war, um das Praxisprojekt durchzuführen. Für 35% war es „Etwas hilfreich“ (n=7) und für 15% sogar „Sehr hilfreich“ (n=3). Nur 10% gaben an, dass das Referenzprojekt „Überhaupt nicht hilfreich“ war, um das Praxisprojekt durchzuführen (n=2). Wobei einer der beiden Teilnehmer, die es nicht hilfreich fanden, angab, kein geeignetes Projekt gefunden zu haben. Der zweite Teilnehmer gab an, das Projekt nicht als Chance wahrgenommen, sondern nur notgedrungen gewählt zu haben - also mit der Absicht, es mit „möglichst wenig zusätzlichem Aufwand zu absolvieren“.

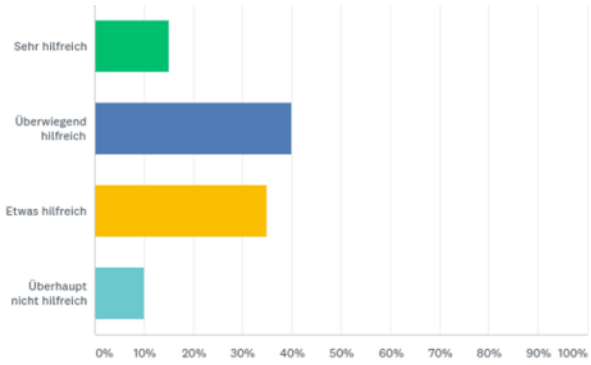


Abb. 4: Referenzprojekt als Hilfe

Ein relativ geringer Teil der Teilnehmer plant, das Referenzprojekt auch in Zukunft zur Strukturierung der Arbeit bzw. ähnlicher Projekte einzusetzen. Ein etwas größerer Teil plant dies nicht. Aber die Mehrheit weiß dies noch nicht. Da das Referenzprojekt für die große Mehrheit „hilfreich“ war, ist dies wahrscheinlich darin begründet, dass die Studierenden zum einen bereits in festen Kontexten in den Unternehmen arbeiten und es unsicher ist, ob die Studierenden in Zukunft in den Themen der Referenzprojekte arbeiten werden oder zum anderen die Referenzprozesse inzwischen beherrschen, so dass diese Hilfe in Zukunft nicht mehr erforderlich ist.

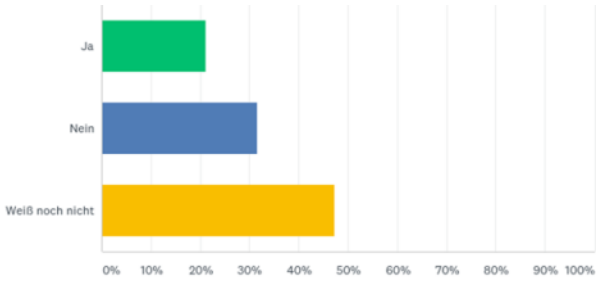


Abb. 5: Einsatz in Zukunft

Zusammengefasst hat sich das Referenzprojekt bei der überwiegenden Mehrheit der Teilnehmer als Orientierungs- und Strukturierungshilfe zur Durchführung der Praxisprojekte bewährt. Es wurde von der überwiegenden Mehrheit der Teilnehmer als wichtige Hilfestellung zur Durchführung des Praxisprojekts eingeschätzt und soll zumindest von einem Teil auch zukünftig als Strukturierungshilfe eingesetzt werden.

Hypothese 3: Das Lernen und Arbeiten in Praxisprojekten führt zu einem Kompetenzerwerb bzw. -zuwachs im Bereich des Tätigkeitsprofils (Berufliche Handlungskompetenz)

Als **Hypothese 3** wurde angenommen, dass sich die Studierenden durch das Lernen und Arbeiten in den Praxis-Projekten die Kompetenzen des Tätigkeitsprofils aneignen.

Die Studierenden sollten beurteilen, „in welchen Bereichen sie im Rahmen des Projektstudiums etwas lernen“ konnten. Es sollten dazu jeweils der Lernertrag in den Bereichen „fachlich“, „methodisch“, „sozial“ und „persönlich“ bewertet werden. Die Mehrheit der Teilnehmer (55%) gab an, im Bereich der fachlichen Kompetenzen „viel“ (50%, n=10) oder „sehr viel“ (5%, n=1) gelernt zu haben. Ähnlich hierzu gab im Bereich der methodischen Kompetenzen die Mehrheit der Teilnehmer (60%) an, „viel“ (50%, n=10) oder „sehr viel“ (10%, n=2) gelernt zu haben. Überraschend ist, dass im Bereich der sozialen Kompetenz kein Teilnehmer angab, „viel“ oder „sehr viel“ gelernt zu haben, sondern alle Teilnehmer angaben, „nichts“ (35%, n=7) oder „wenig“ (65%, n=13) gelernt zu haben. Dies lässt sich ggf. dadurch erklären, dass die Studierenden ja auch ohne das Projektstudium aufgrund ihrer Einbindung in ihr Unternehmen Projekte durchgeführt und Erfahrungen im Bereich der sozialen Kompetenzen gesammelt hätten. Ebenso gaben auch nur 30% der Teilnehmer (n=6) an, „viel“ im Bereich der persönlichen Kompetenzen gelernt zu haben.

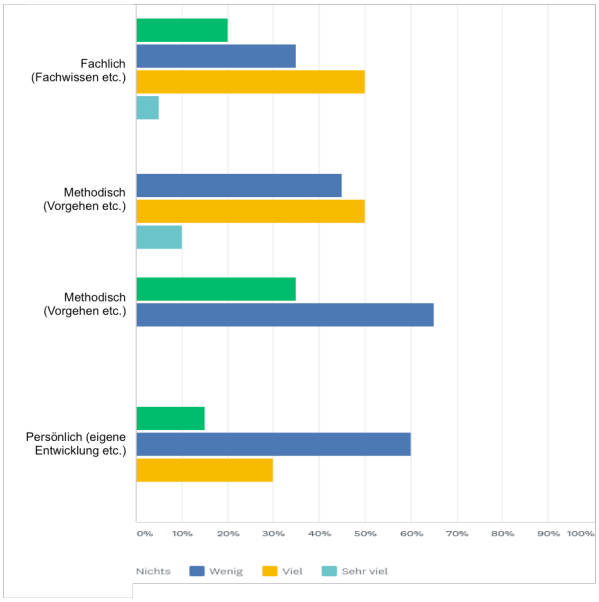


Abb. 6: Lernprozesse in Kompetenzfeldern

Die Studierenden fühlen sich nach Abschluss des Projektstudiums gut auf eine Tätigkeit in dem Tätigkeitsprofil (Data Analyst, IT-Project-Coordinator, IT-Solution-Developer, Software Developer) bzw. auf zukünftige Arbeitsaufgaben vorbereitet. Sie wurden befragt, „inwieweit das Praxisprojekt dazu beigetragen hat, dass die typischen Arbeitsprozesse des Profils jetzt besser beherrscht werden als vorher“. So gibt eine klare Mehrheit (70%) an, dass das Praxisprojekt „sehr“ (10%, n=2) bzw. „überwiegend“ (60%, n=12) dazu beigetragen hat, die typischen Arbeitsprozesse des Profils (Referenzprojekt) jetzt besser zu beherrschen als vorher. Nur eine Minderheit von 30% (n=6) gab an, dass das Praxisprojekt „wenig“ (20%, n=4) oder „nicht“ (10%, n=2) dazu beigetragen hat.

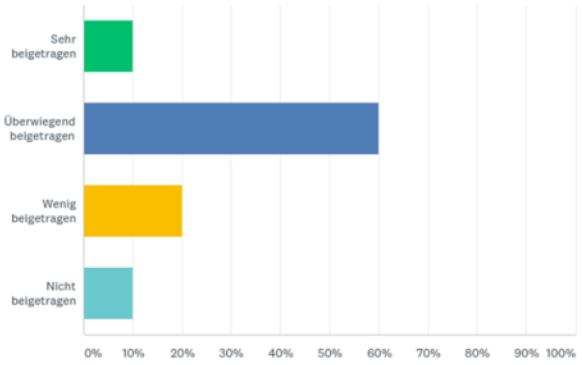


Abb. 7: Vorbereitung auf Tätigkeit

Zusammenfassend führte das Lernen in den Praxisprojekten zu einem Kompetenzerwerb der Teilnehmer im Tätigkeitsprofil des Referenzprojekts. Der Lernerfolg verteilt sich dabei jedoch nicht gleichmäßig auf die verschiedenen Kompetenzbereiche. Während 55% und 60% der Teilnehmer angeben, fachlich bzw. methodisch „viel“ oder „sehr viel“ gelernt zu haben, haben nur 30% im Bereich der persönlichen Kompetenzen „viel“ gelernt und niemand „viel“ oder „sehr viel“ im Bereich der sozialen Kompetenzen. Ggf. spielt die Vorerfahrung der Teilnehmer eine Rolle, welcher in Zukunft bei der Arbeit der Lernprozessbegleiter eine höhere Aufmerksamkeit geschenkt werden sollte.

Hypothese 4: Die Studierenden entwickeln die Fähigkeit, selbständig zu lernen

Als Hypothese 4 wurde angenommen, dass die Studierenden in der Lage waren, selbst gesteuert zu lernen. Hierzu wurde erhoben, inwieweit die Studierenden eigenständig nach Informations- und Wissensquellen suchten, um ihre Wissenslücken zu schließen, sowie inwieweit die Studierenden das Projektstudium als Chance wahrgenommen haben, Neues zu lernen.“

Die Frage, „inwieweit das Projektstudium als Chance gesehen und aktiv genutzt wurde, um etwas Neues zu lernen“, hat die große Mehrheit der Teilnehmer (70%, n=14) positiv beantwortet. 40% der Teilnehmer (n=8) taten dies „Sehr aktiv (bewusst anspruchsvolles und passendes Projekt gesucht)“ und 30% (n=6) „Aktiv (existierendes Projekt gestaltet)“. Lediglich 25% (n=5) machten dies „Notgedrungen (versucht Projekt mit möglichst geringem Aufwand auf das Referenzprojekt abzubilden)“ oder „Gar nicht“ (5%, n=1).

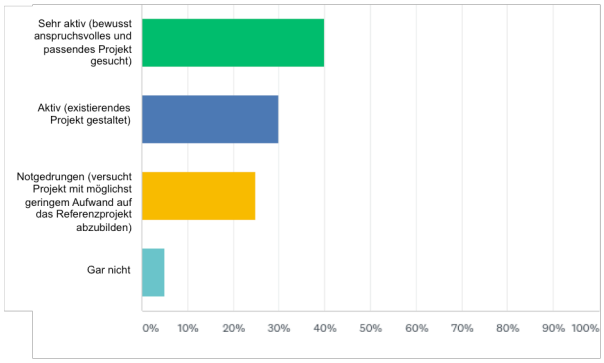


Abb. 8: Projekt als Chance zum Lernen nutzen

Auf die Frage, „wie oft es im Praxisprojekt vorkam, dass selbständig nach fehlenden Informationen gesucht bzw. selbst fehlendes Wissen angeeignet werden musste“, gab die Mehrheit der Teilnehmer (60%, n=12) an, dass dies „häufig“ (35%, n=7) oder „sehr oft“ (25%, n=5) notwendig war. Dies deutet zum einen darauf hin, dass im Praxisprojekt herausfordernde, lernhaltige Tätigkeiten durchgeführt werden mussten (siehe Hypothese 3), und zum anderen,

dass die Studierenden eigenständig Wissenslücken geschlossen und damit gelernt haben. 35% der Teilnehmer (n=7) haben immerhin „Selten“ Wissenslücken selbständig geschlossen. Nur ein Teilnehmer gab an, dass dies „Nie“ erforderlich war (5%, n=1).

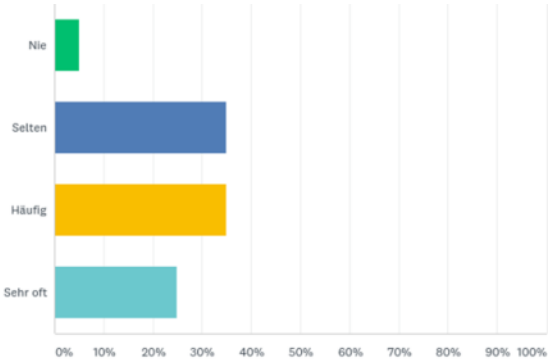


Abb. 9: Wissenslücken selbständig schließen

Die folgenden Zitate stehen beispielhaft für einen hohen eigenständigen Lernertrag:

„...die Bereitstellung der Daten waren weitaus schwieriger und zeitintensiver zu organisieren. Das Wissen hierzu musste ich komplett selbst erarbeiten.“

„Das Projektstudium habe ich genutzt, um mich in das eigene Projektumfeld vertiefend einzuarbeiten und um größtmögliche Synergien zwischen dem "Neu-Gelernten" in den LVs und dem "Alt-Angewandten" im Unternehmen herzustellen.“

Prinzipiell fällt es Studierenden eher schwer, ihre Projektfortschritte und Lernerträge einzuschätzen und zu dokumentieren, vgl. (Platt, 2014). Im Konzept und der Bewertung des Projektstudiums spielt die Reflektion des eigenen Vorgehens und aufgetretener Schlüsselsituationen eine maßgebliche Rolle, bei dem Ziel einen nachhaltigen Wissenserwerb (Lernertrag) zu ermöglichen und zu kontrollieren. Schlüsselsituationen sind solche Situationen, die den Projektverlauf verzögern oder den Projekterfolg verhindern, jedoch keinen inhaltlichen, sondern einen organisatorischen Projektbezug haben, z.B. Terminfindungsprobleme mit Entscheidern, Sprachprobleme, mangelnde Expertise von Beteiligten. Im Projektstudium wird Reflektion daher zu verschiedenen Anlässen ermöglicht und eingefordert. Dazu zählen regelmäßige Reflexionsgespräche, die Zwischenpräsentation, ein Fachgespräch und die Dokumentation am Ende des Projekts.

Auf die Frage, zu welcher dieser Reflexionsanlässe sich die Studierenden der Lernerträge des Praxisprojekts bewusst wurden (Mehrfachnennungen), gab knapp die Hälfte der Studierenden (52,63%, n=10/19) an, dass sie in der Lage waren, auch ohne die Anlässe des Projektstudiums zur Reflektion, sich der Projektfortschritte und Lernerträge des Projektstudiums bewusst zu machen. Die andere Hälfte profitierte von den gebotenen Reflexionsanlässen. Dabei fällt es den Studierenden zu Beginn des Projekts schwerer, Projektfortschritt und Lernerträge durch Reflexionsgespräche selbst einzuschätzen (26,32%, n=6/19). Zentrales Instrument für die Reflektion und die Sicherung der Lernerträge ist die schriftliche Dokumentation (57,89%, n=11/19). Aber auch die Reflektion im Rahmen der Abschlussprüfung (Fachgespräch) zum Ende des Projektstudiums ist nochmal ein wichtiger Anlass zur Reflektion (36,84%, n=7/19).

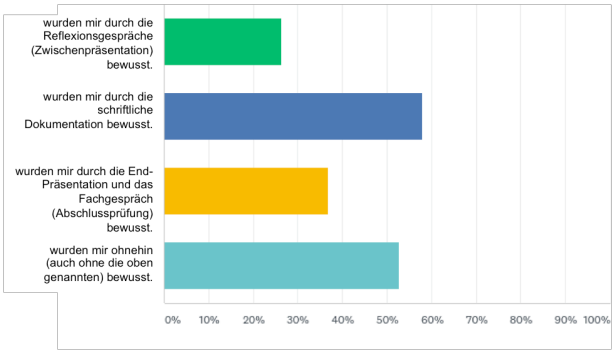


Abb. 10: Reflexionsanlässe zum Bewusstwerden von Lernerträgen

Zusammenfassend kann die Annahme, dass die Teilnehmer in den Praxisprojekten selbständig lernen, prinzipiell bestätigt werden. Zu einem großen Teil initiieren die Teilnehmer lernhaltige Lernsituationen selbst, schließen Wissenslücken eigenständig und sind in der Lage über Reflektion Lernfortschritte einzuschätzen und Lernerträge zu sichern. Andererseits hat ein nicht unerheblicher Anteil der Teilnehmer die Chance zum Lernen nicht aktiv genutzt, kaum die Notwendigkeit zum selbständigen Schließen von Wissenslücken gesehen und wurden auch die Anlässe zur Reflektion nur eingeschränkt als notwendig erachtet. D.h. das Projektstudium bietet die Möglichkeit zum selbständigen Lernen, stellt dieses aber nicht notwendiger Weise sicher, was mit den diversen Freiheitsgraden bei der Auswahl, Durchführung und dem Abschluss des Praxisprojekts zu tun hat. Hier spielen Aspekte der Motivation der Studierenden eine Rolle, wie auch die Sicherstellung lernförderlicher Rahmenbedingungen, die die Auswahl eines geeigneten Projekts maßgeblich beeinflussen. Diese sind bisher im Konzept des Projektstudiums noch wenig berücksichtigt.

Hypothese 5: Die personalen Rollen (Fachberater, Lernprozessbegleiter etc.) sind eine wesentliche Unterstützung des selbstgesteuerten Lernens der Studierenden

Es wurde angenommen, dass Lernprozessbegleiter, Fachberater, Vorgesetzte sowie Kommilitonen und

Kollegen eine wesentliche Unterstützung der selbst gesteuerten Lernprozesse der Studierenden darstellen. Lernprozessbegleiter sind die Dozenten und Dozentinnen des Projektstudiums, die die Studierenden methodisch betreuen, stets als Ansprechpartner zur Verfügung stehen, die Projektdokumentationen bewerten und die Reflexionsgespräche mit den Studierenden durchführen. Die Studierenden sollten jeweils bewerten, „wie wichtig die Unterstützung durch die Lernprozessbegleiter (Projekt-Dozenten) in den verschiedenen Projektphasen war".

Aus den Antworten der Teilnehmer geht hervor, dass die Unterstützung durch die Lernprozessbegleiters wichtig ist für das „Finden eines Praxisprojekts" (77,77% Zustimmung, n=14/19), „Reflektieren des Projekts (Reflexionsgespräche)" (73,33% Zustimmung, n= 11/19) und „Auswerten des Projekts" (68,75% Zustimmung, n=11/19). Weniger wichtig ist diese Hilfe für das „Planen des Praxisprojekts" (46,67% Zustimmung, n=7/19), dem „Bearbeiten des Projekts" (37,5% Zustimmung, n=6/19) sowie der „Dokumentation des Projekts" (37,5% Zustimmung, n=6/19).

Aus den Kommentaren der Teilnehmer geht zudem hervor, dass ein Mentor (Rolle des Vorgesetzten) aus dem Unternehmen gewünscht wird, der die Studierenden beim Finden eines geeigneten Projekts unterstützt und ihnen bei der Bearbeitung den Rücken freihält: *„Es wäre sicher hilfreich, wenn von Beginn an JEDER Student einen unternehmensinternen Mentor (z.B. der Manager, der Kollege mit Erfahrung zum Studium, etc.) an der Seite hat, der über die Lerninhalte des Studiums im Bilde ist und eventuell proaktiv zur Vorbereitung (z.B. Themensuche) beitragen und den er fragen kann."*

Aus den Kommentaren der Teilnehmer wird ebenfalls die Bedeutung der Unterstützung der Studierenden über den Austausch der Studierenden untereinander deutlich. Aus den Kommentaren geht auch hervor, dass mehr Präsenz im Projektstudium und eine Intensivierung des Austauschs unter den Kommilitonen gewünscht wird: *„Die Kommilitonen waren sehr zurückhaltend und ich glaube, es wäre ein Mehrwert gewesen, hätten wir gewusst, wer was bearbeitet." sowie „Öfter eine Präsenzveranstaltung!" sowie „Ich finde es auch sinnvoll, wenn man nach Projektabschluss die Ergebnisse im eigenen Kurs präsentieren würde. Das Interessante ist ja nicht nur die „Abarbeitung" der Methodik, sondern der Einblick in andere Themen, Ansätze und Unternehmen. Dies würde auch den Austausch im Studiengang fördern."*

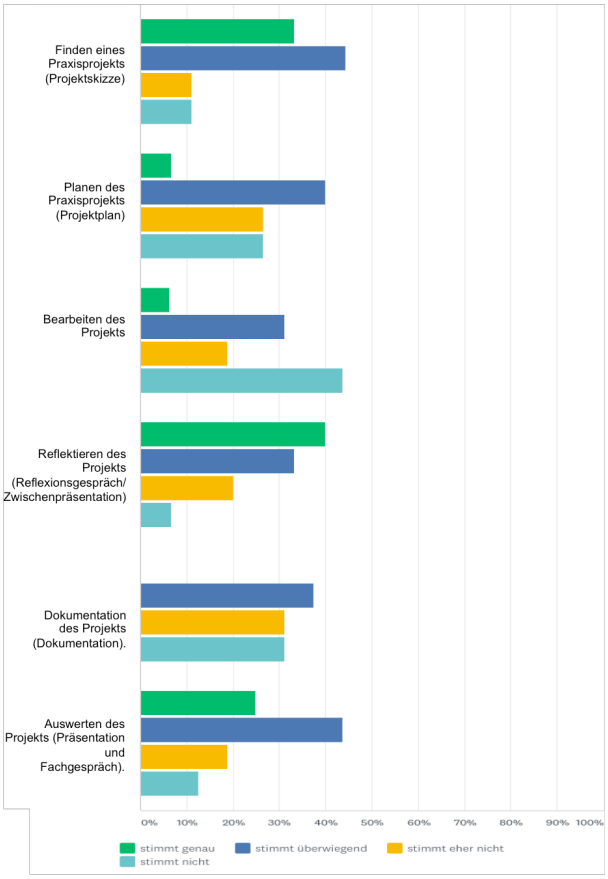


Abb. 11: Unterstützung durch Lernprozessbegleiter

Zusammenfassend zeigen die Ergebnisse, dass eine Unterstützung durch einen Lernprozessbegleiter über das ganze Projekt hinweg, also am Anfang (Projektfindung), während des Projekts (Reflektion) und auch am Ende des Projekts (Auswerten) wichtig ist, um Lernerträge zu identifizieren und zu sichern. Dabei ist diese Unterstützung nicht für alle Tätigkeitsbereiche gleichermaßen wichtig. Es gibt auch Phasen bzw. Tätigkeiten, bei denen eine fachliche Unterstützung hilfreich ist. Zudem ist für bestimmte Aspekte auch die Unterstützung durch einen Mentor (Vorgesetzter) und der Austausch unter den Studierenden wichtig.

Hypothese 6: Unterstützung des selbstgesteuerten Lernens der Studierenden durch IT-Unterstützung (Medien bzw. E-Learning-Umgebung)

Als **Hypothese 6** wurde angenommen, dass die E-Learning-Umgebung (moodle) eine wichtige Unterstützung für das Lernen der Teilnehmer darstellt. Hierzu wurde gefragt, „ob sich die Teilnehmer mehr Informationen oder Austausch über die Lernplattform (moodle) gewünscht hätten".

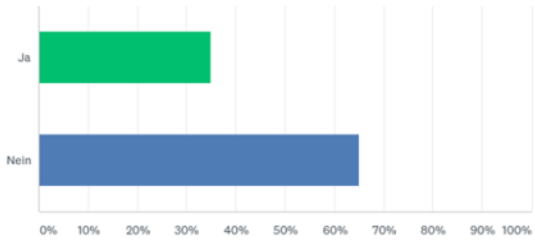


Abb. 12: Nutzung der Lernplattform

Nur ca. ein Drittel der Teilnehmer (35%, n=7) gab an, dass sie sich mehr Informationen oder Austausch über die Lernplattform (moodle) gewünscht hätten. Die deutliche Mehrheit (65%, n=13) hat diesen Wunsch nicht.

Bei der Auswertung der Kommentare wird deutlich, dass die Lernplattform kaum für den individuellen Wissenserwerb zur fachlichen Bewältigung des Projekts genutzt wurde (klassisches E-Learning). Vielmehr war die Plattform wichtig für a) die Organisation der Lehrveranstaltung, b) den Austausch mit den Dozenten und c) den Austausch mit den anderen Studierenden. *Zu a) „Frühere Abgabe des Projektes sollte ermöglicht werden.“, „Informationen zur geplanten Herangehensweise“, „Beispiele wären hilfreich gewesen“, b) „Da die Dozenten versuchen, ALLE Fragen auch auf die Moodle-Plattform (also für alle) zu bringen, ist man über die wichtigsten Punkte meist informiert. Das sollte auch so beibehalten werden.“ zu c) „Eventuell Vorstellung der Themen anderer Kommilitonen, um ähnliche Problematik zu identifizieren und Herangehensweise diskutieren zu können.“.*

Zusammenfassend lässt sich festhalten, dass eine E-Learning-Plattform im klassischen Sinne nicht erforderlich ist, da die Studierenden sich das erforderliche Wissen selbständig aus Quellen im Unternehmen (Kollegen, Bücher etc.) oder im WWW beschaffen. Wichtig ist aber eine Online-Plattform für die formale Organisation der Lehrveranstaltung, aber vor allem für den Austausch mit den Dozenten sowie zwischen den Studierenden untereinander.

Hypothese 7: Unterstützung des selbstgesteuerten Lernens der Studis durch fachliche Fundierung

Als **Hypothese 7** wurde angenommen, dass die klassische Lehrveranstaltung zum Themengebiet des Praxisprojekts im vorherigen Semester eine wichtige fachliche Unterstützung für das Lernen und die Bewältigung des Praxisprojekts der Teilnehmer darstellt. Hierzu wurde die Teilnehmer, die das Profil „Data-Analyst/-in“ gewählt hatten, befragt, „inwieweit die angebotene, klassische Lehrveranstaltung (Data Analytics) bei der Bewältigung des Projekts geholfen hat“.

Dabei gaben alle 12 Studierenden, die das Profil „Data-Analyst/-in“ gewählt hatten (100%, n=12), an, dass sie im Rahmen der klassischen Lehrveranstaltung überwiegend Dinge gelernt haben, die für das Praxisprojekt wichtig und nützlich waren. Keiner der Teilnehmer (n=0) gab an, dass sie das Praxisprojekt auch ohne die vorherige Lehrveranstaltung hätten erfolgreich bewältigen können. Interessant ist, dass 15% (n=3) der Teilnehmer aussagten, dass aufgrund der klassischen Lehrveranstaltung das Projektstudium zu dem Thema „Data Analytics“ nicht mehr erforderlich ist, um Probleme in der Praxis selbständig zu lösen. Es muss sich also um eine sehr gute, praxisorientierte Lehrveranstaltung gehandelt haben.

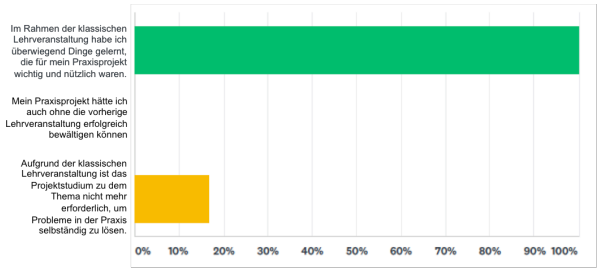


Abb. 13: Fachliche Fundierung „Data Analytics“

Zusammenfassend scheint die vorhergehende klassische Lehrveranstaltung eine große Bedeutung für den Erfolg des Praxisprojekts zu haben. Darin ist wahrscheinlich auch die geringe Bedeutung eines „Fachberaters“ als unterstützende personale Rolle begründet.

Hypothese 8: Akzeptanz des Projektstudiums

Als **Hypothese 8** wurde angenommen, dass das Projektstudium für die Studierenden eine attraktive Form der Lehre darstellt (Akzeptanz-Hypothese). Hierzu sollte jeweils bewertet werden, „inwieweit ein persönlicher Gewinn aus dem Projektstudium gezogen wurde“, „worin ungünstige Bedingungen bestanden“ und „was in Zukunft verbessert werden könnte“ sowie, ob „es leicht fiel, die Dokumentation zu erstellen“.

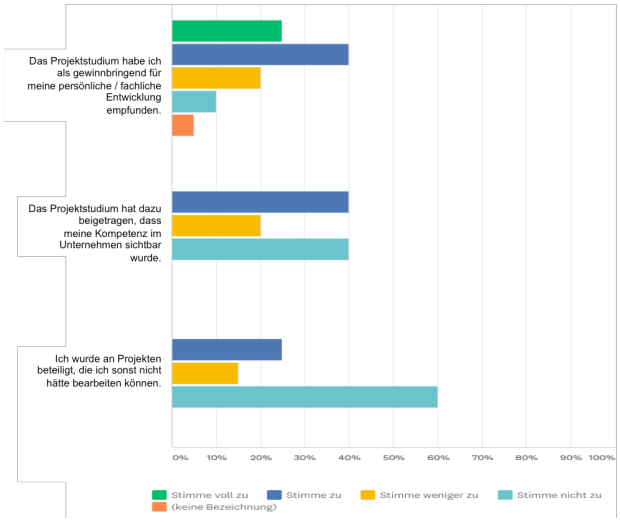


Abb. 14: Persönlicher Gewinn

Die große Mehrheit der Teilnehmer (65%, n= 13) gab an, dass das Projektstudium als gewinnbringend für die persönliche und fachliche Entwicklung empfunden wurde. 40% (n=8) gaben an, dass das Projektstudium dazu beigetrage hat, dass die eigenen Kompetenzen im Unternehmen sichtbar wurden. Zudem gaben 25% (n=5) an, dass sie an Projekten beteiligt wurden, die sie sonst nicht hätten bearbeiten können.

Als „ungünstige Bedingungen in den Unternehmen, die die Arbeit am Praxisprojekt beeinträchtigten“, wurden vor allem ausgewählt (Mehrfachnennungen möglich):

- Nicht genug Zeit verfügbar (61,11%, n=11/18)
- Schwer in meinem Unternehmen ein passendes Projekt zu finden (55,56%, n=10/18)
- Praxisprojekt nicht im gewohnten Arbeitsumfeld möglich (38,89%, n=7/18)

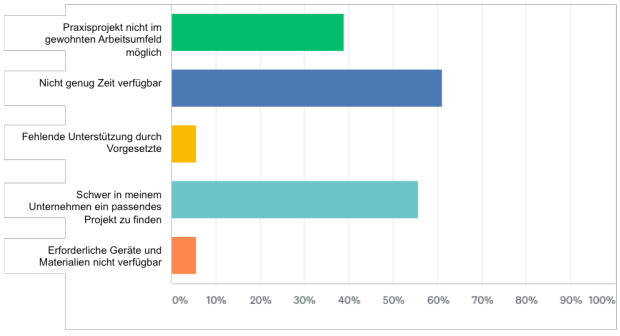


Abb. 15: Ungünstige Bedingungen

Fehlende Unterstützung durch Vorgesetzte (5,56%, n=1/18) und erforderliche Geräte und Materialien nicht verfügbar (5,56%, n=1/18) werden dabei kaum als Problem genannt.

Aus den Kommentaren der Teilnehmer geht zusätzlich hervor, dass es schwierig ist, Projekte zu finden, die zeitlich genau in ein Semester passen (Semesterrhythmus, d.h. Start-/End-Termin und Dauer). Zudem kann es gerade bei großen Unternehmen mit länger laufenden Projekten schwierig sein, jedes Semester ein geeignetes Projekt zu finden, wenn diese Projekte keine Tätigkeiten für die unterschiedlichen Rollen des Projektstudiums beinhalten. Andererseits gab es aber bereits Beispiele, in denen die Studierenden im gleichen Unternehmensprojekt im ersten Semester in der Rolle des IT Solution Developers und im zweiten Semester in der Rolle des Data Analysten agierten.

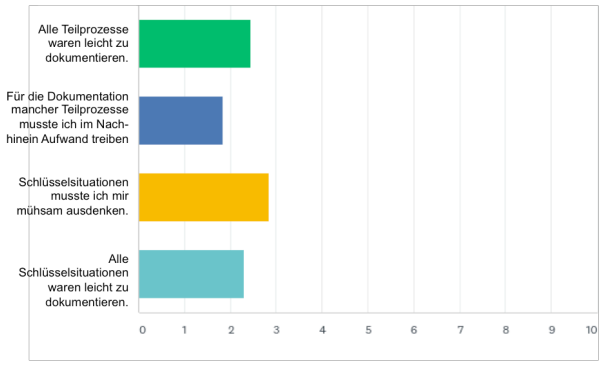


Abb. 16: Aufwand für Dokumentation

Zum Thema Dokumentation der Teilprozesse gab nur eine knappe Mehrheit von 52% (n=10/19) an, dass die Teilprozesse leicht zu dokumentieren seien (Mittelwert: 2,37). Dagegen gab eine deutliche Mehrheit an (84%, n=16/19) an, dass für die Dokumentation mancher Teilprozesse ich im Nachhinein Aufwand betrieben werden musste (Mittelwert: 1,89). Bezüglich der Dokumentation der Schlüsselsituationen gaben nur 26% (n=5/19) an, dass sie sich die Schlüsselsituationen mühsam ausdenken mussten (Mittelwert: 2,89), und 68% (n=13/19) gaben an, dass alle Schlüsselsituationen leicht zu dokumentieren waren (Mittelwert: 2,26).

Zusammenfassend lässt sich feststellen, dass für über die Hälfte der Studierenden das Lernen unter erschwerten Bedingungen stattfand, weil die Arbeit im Praxisprojekt unter ungünstigen Lern-Bedingungen im Unternehmen stattfand. Insbesondere Zeitmangel und hoher Projektdruck waren hierfür die Ursache. Die größte organisatorische Herausforderung besteht im Finden eines geeigneten Praxisprojekts im Unternehmen, das sowohl zum Referenzprojekt (Profil) als auch zum Zeitplan des Semesters passt. Zudem wurde die Aufgabe des Dokumentierens der Teilprozesse von der Mehrzahl der Studierenden als schwierig und aufwendig empfunden.

Schlussfolgerungen

Aus den Ergebnissen der Evaluation sind folgende Schlüsse zu ziehen:

Schlussfolgerungen für das Konzept des Projektstudiums

Das Konzept des Arbeitsprozess-integrierten Projektstudiums hat sich prinzipiell bewährt:

Die Referenzprojekte stellen für die Mehrheit der Teilnehmer eine Orientierungshilfe zur Projektfindung (Hypothese 1) und eine Strukturierungshilfe für das Lernen (Hypothese 2) dar. Das Projektstudium führt zu einem Kompetenzerwerb im Bereich des Tätigkeitsprofils (Hypothese 3) und die Studierenden entwickeln die Fähigkeit zum selbständigen Lernen (Hypothese 4). Die Unterstützung des selbständigen Lernens durch personale Rollen, insbesondere einen Lernprozessbegleiter, ist sehr wichtig (Hypothese 5). Die Unterstützung des Lernens durch eine IT-Umgebung (E-Learning) ist dabei weniger für die Vermittlung von Fachinhalten, aber für die formale Organisation und den Austausch zwischen den Beteiligten wichtig (Hypothese 6). Zudem ist die vorbereitende klassische Lehrveranstaltung von großer Bedeutung für den Erfolg (Hypothese 7).

Alle Studierenden haben das Projektstudium erfolgreich bewältigt. Für sie war das Projektstudium durchgängig ein Erfolg. Die Mehrheit der Teilnehmer sieht es als Chance, Neues zu erlernen und nutzt es dafür aktiv. Darüber hinaus wird es von einer Mehrheit als gewinnbringend für die persönliche und fachliche Entwicklung empfunden wurde. Das Projektstudium hat sogar dabei geholfen, dass die eigenen Kompetenzen im Unternehmen sichtbar wurden und dass Studierende an Projekten beteiligt wurden, die sie sonst nicht hätten bearbeiten können (Hypothese 8).

Die Evaluation legt nahe, dass das Konzept des Arbeitsprozess-integrierten Projektstudiums durchführbar ist, für die Studierenden einen Erfolg darstellt und den gewünschten Kompetenzerwerb ermöglicht.

Optimierungspotenzial für das Projektstudium

Ohne die obige Einschätzung relativieren zu wollen, zeigen die Ergebnisse Möglichkeiten der Verbesserung sowohl des Konzepts als auch der praktischen Umsetzung des Projektstudiums auf:

- Die größte Herausforderung für die Studierenden besteht darin, ein zum Referenzprojekt und Semesterzeitplan passendes Praxisprojekt in ihrem jeweiligen Unternehmen zu finden. Hierzu wurde bereits das Projektstudium zeitlich von der begleitenden bzw. vorbereitenden klassischen Lehrveranstaltung entzerrt (siehe unten). Es ist aber eine weitergehende Unterstützung erforderlich. Bspw. könnte das Spektrum möglicher Referenzprojekte erweitert werden (z.B. Requirements Engineer/-in), passende Projekte in anderen Unternehmen bereitgestellt werden oder die Durchführung über verschiedene Semester ermöglicht werden.
- Ein Lernerfolg konnte zwar – insbesondere im Bereich der fachlichen und methodischen Kompetenzen – nachgewiesen werden, aber dieser verteilt sich nicht gleichmäßig auf die verschiedenen Kompetenzbereiche. Insbesondere die Stärkung der sozialen und personalen Kompetenzen wird nicht erkannt oder findet im Rahmen des Projektstudiums zu wenig statt. Daraus ergeben sich zwei Schlussfolgerungen:
  - Erstens muss die bewusste Reflektion über personale und soziale Kompetenzen verbessert und gefördert werden. Dies erfolgt derzeit implizit über die Dokumentation von 10 Schlüsselsituationen (Problem, Lösung, Lernertrag). Dies könnte durch die Pflicht zur Dokumentation von Lernerträgen explizit zu den verschiedenen Kompetenzbereichen erreicht werden.
  - Zweitens muss das Lernen im Bereich der sozialen und personalen Kompetenzen explizit gefördert und gefordert werden. Dazu bieten sich insbesondere die Förderung des Lernens in den Bereichen der Kommunikation, des Verhandelns und des Führens an, um den Studienzielen des Masterstudiengangs „Professional IT-Business“ zu entsprechen. Hierzu bedarf es einer Erweiterung des Konzeptes der Arbeitsprozess-integrierten Projektstudium mit Fokus auf die Aneignung und Stärkung sozialer Kompetenzen.
- In diesem Zusammenhang sehen wir weiteres Optimierungspotential in der Verbesserung des Reflexionsprozesses. Insbesondere bei der Dokumentation fällt auf, dass die Studierenden oftmals trotz der gezielten Fragen eher über das Projekt berichten, anstatt zu getroffenen Entscheidungen oder Schlüsselsituationen tatsächlich reflektieren (Lernertrag). Es scheint, als würden die Studierenden nur zu den geforderten zwei Zeitpunkten, nämlich zur Zwischenpräsentation und zur Erstellung der Dokumentation für die Endpräsentation über ihre Schlüsselsituationen reflektieren. Dieser zeitliche Abstand zum Erlebten führt jedoch dazu, dass die Studierenden eher künstliche, nur scheinbar die Form wahrende Reflektionen formulieren. Um näher am Prozess und den erlebten Situationen zu sein, müssten die Studierenden tatsächlich, wie eigentlich auch gefordert, ihre Dokumentation über die gesamte Projektzeit schreiben und

pflegen. Dies wäre ganz im Sinne eines *reflective journals* vgl. (Prior, 2016), als welches die Dokumentation gedacht ist. Eventuell müssen dazu weitere Prüfpunkte vereinbart werden, um die Kontinuität der Reflektionen zu verbessern.

- Bei den Studierenden zeigten sich unterschiedliche Grade an Selbststeuerung. Zwar haben die Studierenden aufgrund der vorbereitenden klassischen Lehrveranstaltung eine bestimmte fachliche Vorerfahrung, doch diese kann dennoch aufgrund des beruflichen Einsatzgebietes immer noch recht unterschiedlich sein. So benötigen einige Studierenden eher Unterstützung bei der Bewältigung herausfordernder Aufgaben, während andere Studierende (mit umfassender Vorerfahrung) eher Hilfe beim Finden oder Initiieren lernhaltiger Situationen bzw. gar Projekte benötigen. Die fachliche Vorerfahrung der Studierenden sowie die diversen Freiheitsgrade bei der Auswahl und Durchführung des Praxisprojekts müssen im Konzept und bei der Arbeit der Lernprozessbegleiter in Zukunft stärker berücksichtigt werden. Dies betrifft insbesondere die Schaffung lernförderlicher Rahmenbedingung als auch die Motivation der Studierenden, die wesentlich die Auswahl von Projekten beeinflussen.
- Ein wesentliches Optimierungspotenzial besteht darin, jedem Studierenden einen „Mentor" aus dem Unternehmen explizit zuzuweisen, der sowohl fachlich als auch organisatorisch unterstützt. Zwar gibt es einen „Fachberater" in der Hochschule und die Rolle des „Vorgesetzten" im Unternehmen sowie auch Kollegen, die fachlich und organisatorisch unterstützen sollen. Allerdings ist diese Unterstützung bisher implizit, d.h. nicht formal festgelegt.
- Wie die Analyse der IT-Unterstützung gezeigt hat, sehen die Studierenden einen großen Bedarf in der Unterstützung des Austauschs untereinander. Daher sollte auch der Austausch unter den Studierenden gefördert werden, um als „Learning Community" einen gegenseitigen Mehrwert zu schaffen.

Zusammenfassung und Ausblick

Die Ergebnisse der Umfrage legt nahe, dass sich das Konzept und die praktische Umsetzung des Arbeitsprozess-integrierten Lernens im Projektstudium prinzipiell bewährt hat. Das Arbeitsprozess-integrierte Projektstudium ist praxisorientiert, da reale Projekte in Unternehmen durchgeführt werden. Es ist prozessorientiert, da das Lernen in und entlang von Arbeitsprozessen erfolgt. Es ist erfahrungsgeleitet, denn die Projektarbeit selbst ist Lern- und Reflektionsgegenstand. Aufgrund der häufigen Reflektion in Zwischen- und End-Präsentation sowie der Dokumentation ist das Projektstudium reflektiert und es ist Lerner-orientiert, da die Inhalte und die Arbeitsabläufe aufgrund des individuellen Projektes individualisiert sind. Der gesamte Lern- und Arbeitsprozess läuft selbstorganisiert mit Unterstützung von Fachberatern und Lernprozessbegleitern ab.

Die hier berichtete Umfrage belegt, dass die Referenzprojekte eine gute Orientierungshilfe zur Projektfindung und eine gute Strukturierungshilfe für das Lernen darstellen. Die Mehrzahl der Studierenden bezeichnet das Projektstudium als persönlichen Gewinn. Zudem wird mehrheitlich angegeben, dass die für die jeweiligen Rollen typischen Arbeitsprozesse nach Absolvieren des Projektstudiums besser beherrscht werden. Die Mehrheit hat das Projektstudium als Chance gesehen und auch aktiv genutzt, um Neues zu erlernen.

Wesentliche Erkenntnisse hat die Umfrage über die Reflektion zur Kompetenzgewinnung offenbart. Während noch eine Mehrzahl einen methodischen und/oder fachlichen Kompetenzgewinn angibt, reflektieren die Teilnehmer kaum persönlichen und gar keinen sozialen Kompetenzgewinn. Hierzu muss eine Anpassung des Konzeptes dahingehend erfolgen, dass die Studierenden soziale Kompetenz bewusst erlernen und darüber reflektieren. Bereits in den kommenden Durchgängen des Projektstudiums sollen die sozialen Kompetenzen besonders in den Bereichen der Kommunikation, des Verhandelns und des Führens gestärkt werden. Entsprechende Erweiterungen und Optimierungen des Konzeptes sind gegenwärtig in Arbeit und werden in einem zukünftigen Beitrag Berichtsgegenstand.

Literatur

Brand, T. (2014): Evaluation einer arbeitsprozessorientierten IT-Weiterbildung: „IT-Spezialisten". Dissertation, Erfurt, Universität Erfurt.

Buhr, R.; Freitag, W.; Hartmann, E.A.; Loroff, C.; Minks, K.H.; Mucke, K.; Stamm-Riemer, I. (2008) (Hrsg.): Durchlässigkeit gestalten! Wege zwischen beruflicher und hochschulischer Bildung. Münster, Waxmann.

Dingsoer, T. (2005): Postmortem reviews: purpose and approaches in software engineering. Information a. Software Technology, 47 (5), 293-303.

Freitag, W.; Hartmann, E.A.; Loroff, C.; Stamm-Riemer, I.; Volk, D.; Buhr, R. (2011) (Hrsg.): Gestaltungsfeld Anrechnung - Hochschulische und berufliche Bildung im Wandel. Münster, Waxmann.

Fuchs-Kittowski, F.; Freiheit, J.; Siegeris, J. (2017): Arbeitsprozess-integriertes Projekt-Studium in Unternehmen. In: Tagungsband des 15.

Workshops "Software Engineering im Unterricht der Hochschulen" 2017, 41-50

Götz, K. (1993): Zur Evaluierung beruflicher Weiterbildung. Deutscher Studienverlag, Weinheim.

Mattauch, W.; Kubath, S. (2005): Evaluation der arbeitsprozessorientierten Weiterbildung. ISST-Bericht 74/05, Berlin, Fraunhofer ISST.

Platt, L. (2014): The 'wicked problem' of reflective practice: a critical literature review, Innovations in Practice 9 (1).

Prior, JR, Arjpru, S; Leaney, JR (2014): 'Towards an industry-collaborative, reflective software learning and development environment. In: Proc. of the 23rd Australasian Software Engineering Conf. ASWEC 2014, IEEE, Sydney, Australia.

Prior, JR.; Ferguson, S.; Leaney, J. (2016): Reflection is hard: teaching and learning reflective practice in a software studio. In: Proc. of the Australasian Computer Science Week ACSW '16, Canberra, Australia, 2016, 1–8.

Schön, DA. (1983): The reflective practitioner: how professionals think in action. New York: Basic Books. ISBN 046506874X. OCLC 8709452.

Upchurch, R.; Sims-Knight, J. (1999): Reflective Essays in Software Engineering. In: Proc. of 29th ASEE/IEEE Frontiers in Education Conference, IEEE, San Juan, Puerto Rico.

Wottawa, H.; Thierau, H. (1990): Handbuch Evaluation. Bern: Verlag Hans Huber.

Lernzieldefinitionen für Software Engineering — the Good, the Bad and the Ugly

Axel Böttcher, Veronika Thurner, Olga Nikoliai
Fakultät für Informatik und Mathematik, Hochschule München
<vorname>.<nachname>@hm.edu

Zusammenfassung

Spätestens seit der Bologna-Reform sind wir als Lehrende immer wieder dazu angehalten, Modulbeschreibungen zu verfassen oder zu überarbeiten und in diesen kompetenzorientierte Lernziele zu definieren.

Um die Qualitätssicherung von Lernzieldefinitionen zu unterstützen hatten wir zunächst die Absicht, ein Softwarewerkzeug zur automatisierten Qualitätsprüfung von Lernzieldefinitionen zu realisieren. Dabei haben wir schnell festgestellt, dass vor einer Automatisierung noch viele offene Fragen auf konzeptueller Ebene zu klären sind, hinsichtlich Kriterien und Indikatoren für die Qualität von Lernzieldefinitionen. Mit dieser Arbeit wollen wir eine Diskussion der Erkenntnisse anstoßen, die wir bei der Untersuchung dieser Fragen gewonnenen haben.

Ausgehend von der Beobachtung, dass unsere Curricula aus den verschiedenen Perspektiven von Lehrenden, Lernenden, Akkreditierungsagenturen sowie Arbeitgebern betrachtet werden, formulieren wir zunächst Qualitätskriterien für kompetenzorientierte Lernzielbeschreibungen.

Auf Basis dieser Kriterien haben wir systematisch Beschreibungen für Software-bezogene Module analysiert. Anhand von diesen empirischen Befunden identifizieren wir Indikatoren für positive (*good*) und weniger positive (*ugly* bis *bad*) Lernzielbeschreibungen. Diese abstrahieren wir anschließend zu allgemeineren Kriterien, benennen das jeweils zugrundeliegende Prinzip und gewichten diese Kriterien sodann nach ihrer Bedeutung für die Qualität einer Lernziedefinition.

Gute Lernzieldefinitionen beschreiben das gewünschte Lernergebnis kompetenzorientiert, d. h. als eine Kombination aus Inhalt und Handlungskomponente, die in Form von beobachtbarem Verhalten in Studierenden-zentrierter Form formuliert ist. Sie sind dabei in ihrer Bedeutung verständlich, eindeutig und auf den Punkt.

Eine erste empirische Analyse von bestehenden Modulbeschreibungen zeigt, dass diesbezüglich noch vielfältiges Verbesserungspotenzial vorhanden ist.

Abstract

At least since the Bologna reform, we the lecturers are expected to compose or refine module descriptions which comprise the definition of competence oriented learning objectives.

To support the quality assurance of definitions of learning objectives, our first intention was to develop a software tool that automatically quality checks definitions of learning objectives. However, we quickly realized that prior to an automization, a host of still open questions has to be clarified on a conceptual level, with regard to criteria and indicators for the quality of learning objectives. This work is intended to initiate a discussion of our findings.

Starting from the observation that our curricula are viewed from the different perspectives of lecturers, learners, accreditation organizations and employers, we specify quality criteria for competence oriented learning objectives as a first step.

As a second step, we systematically analyzed the descriptions of software related modules. Based on these empirical results, we identify indicators for positive (*good*) and less positive (*ugly* to *bad*) definitions of learning objectives. Subsequently, we generalize these into more abstract criteria, name the respective underlying principle and appraise their relevance for the quality of a definition of learning objectives.

Well formulated definitions of learning objectives describe the intended learning outcome in a competence oriented way, i. e., as a combination of content and action, which is expressed as an observable behaviour and in a student centered way. As well, good learning objectives are easy to understand, unambiguous and to the point.

A first empirical analysis of existing module descriptions indicates that there is still lots of room for improvements in this area.

1 Motivation

Spätestens seit der Bologna-Reform steht fest, dass für nahezu alle an europäischen Hochschulen durchgeführten Lehrveranstaltungen als Grundlage eine Modulbeschreibung zu erstellen ist. In dieser ist unter anderem zu definieren, welche Lernergebnisse dieses Modul anstrebt, d. h. in den Studierenden entwickeln soll. Diese Lernergebnisse sind auf kompetenzorientierte Weise zu beschreiben.

Gemäß der Definition von Weinert (Weinert, 2001, S. 27) verstehen wir unter Kompetenzen „die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen, volitionalen und sozialen Bereitschaften und Fähigkeiten, um die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können“. Kompetenzen drücken also ein Handlungspotenzial eines Individuums in einer bestimmten Situation aus, d. h. die Fähigkeit, bei Bedarf etwas Bestimmtes zu tun. Sie umfassen immer sowohl eine inhaltliche als auch eine prozessuale Komponente (UZH, 2008). Der Inhalt bestimmt, um was es dabei eigentlich geht, beschreibt somit den Gegenstand, auf den sich die Kompetenz bezieht. Die Handlungskomponente dagegen legt die Art der Tätigkeit fest, die auf bzw. mit dem Inhalt durchzuführen ist, repräsentiert also das Kompetenzniveau, d. h. den zu erreichenden Grad an Handlungsbefähigung.

Auch in der Vor-Bologna-Ära wurde an deutschen Hochschulen das verfügbare Lehrangebot dokumentiert, ehemals in Form von gedruckten Vorlesungsverzeichnissen. Ähnlich wie die heutigen Modulbeschreibungen sollten auch jene Urversionen eine Vorstellung davon vermitteln, was die angehenden Hörerinnen und Hörer in der Vorlesung, bzw. dem Modul, erwartet. In der Regel fokussierten diese Beschreibungen einen Fachbegriff-reichen Titel, der im Idealfall noch konkretisiert wurde durch eine Liste von zu behandelnden Inhalten, jedoch meist ohne ergänzende Handlungskomponente.

Um die Bologna-Vorgaben zu erfüllen mussten diese rudimentären Inhaltsangaben der historischen Modulbeschreibungen zu kompetenzorientierten Lernzieldefinitionen ausgebaut werden. Dieser Prozess war mit diversen Stolpersteinen unterschiedlicher Art durchsetzt, rangierend von psychologischen („Wozu soll DAS denn gut sein?“) bis hin zu handwerklichen („Kompetenz ist halt, wenn ich den Inhalt kann, das ist doch klar!“). Die ersten Ergebnisse dieser Bemühungen wirkten gelegentlich eher wie das Resultat einer ungeliebten, aber verpflichtenden Fingerübung, blieben dabei in ihrem Nutzen jedoch wenig erhellend.

Mittlerweile hat sich die Erkenntnis etabliert, dass in Zeiten der universellen Verfügbarkeit von Faktenwissen über das Internet der Erwerb dieses Faktenwissens für sich alleine als Lernziel einer akademischen Ausbildung keinesfalls ausreichend ist. Vielmehr fordern wir von unseren Studierenden, dass sie Fähigkeiten und Fertigkeiten entwickeln, die zwar auf Faktenwissen basieren, aber in ihrer Handlungsdimension weit über das reine Wiedergeben dieses Faktenwissens hinaus gehen. Nur wenn die Studierenden die faktischen Inhalte zielgerichtet und problemlösungsorientiert einsetzen und dabei die Qualität der verwendeten Vorgehensweise ebenso wie der erzielten Ergebnisse treffend beurteilen können, sind sie auf die Anforderungen des Arbeitsmarktes angemessen vorbereitet. Entsprechend müssen kompetenzorientierte Lernzieldefinitionen die geforderten Fähigkeiten und Fertigkeiten auf handlungsorientierte Weise so formulieren, dass die unterschiedlichen Expertise-Grade darin erkennbar werden.

Des Weiteren wurde mit dem Grundgedanken des Constructive Alignment (Biggs u. Tang, 2011) die Sinnhaftigkeit kompetenzorientierter Lernzieldefinitionen untermauert, sowohl auf der psychologischen als auch auf der handwerklichen Ebene. Eine klare Zielvorstellung zu haben ist für uns als Lehrende die Voraussetzung dafür, passende Lehr-Lern-Methoden auszuwählen und einzusetzen, die möglichst systematisch auf das Erreichen des gesetzten Lernzieles hinwirken. Analog dazu ist die Prüfung so zu gestalten, dass sie möglichst fokussiert das Erreichen der Lernziele überprüft. Beides setzt ein klares Verständnis der Lernziele voraus.

Halten wir also fest, dass die kompetenzorientierte Lernzieldefinition notwendiger und sinnvoller Bestandteil einer jeden Modulbeschreibung ist, die vielfältigen Nutzen stiftet. Trotz der immensen Wichtigkeit kompetenzorientierter Lernzieldefinitionen sind die real existierenden Modulbeschreibungen, die man so findet, jedoch qualitativ sehr unterschiedlich – the Good, the Bad and the Ugly.

Hier besteht also weiterhin Handlungsbedarf, denn nur qualitativ gute Lernzieldefinitionen führen auch zu dem erhofften Nutzen.

2 Stand der Praxis

Ein Lernziel beschreibt das am Ende eines Lernprozesses erwünschte Ergebnis derart, dass nachprüfbar ist, ob dieses erwünschte Ergebnis auch tatsächlich erreicht wurde (Terhart, 2005, S. 111f). Lernziele beschreiben also möglichst exakt die angestrebten Lernergebnisse (Arnold u. a., 1999, S. 79).

Darüber, ob ein definiertes Lernziel ein Versprechen darstellt oder nicht, herrscht in der Praxis eine gewisse Uneinigkeit. Häufig sind Lernziele so formuliert, als würde keinerlei Zweifel daran bestehen, dass sie auch erreicht werden, wie beispielsweise „Sie als Teilnehmende durchdringen das Prüfungsrecht aus Sicht des Prüfers und der Studierenden. Sie wenden es in der Prüfungssituation sicher an“ (DiZ, 2018).

De facto machen jedoch die meisten Lehrenden und auch viele Lernende früher oder später die Erfahrung, dass eben *nicht* alle Teilnehmenden auch

die gewünschten Ziele in vollem Umfang erreichen. Entsprechend sind in der Praxis die angestrebten Lernziele also nicht zwangsweise deckungsgleich mit den tatsächlich erreichten Lernergebnissen (Bischoff u. a., 2017). In einer Handreichung der HRK zur Formulierung von Lernzielen schlägt (Gröblinghoff, 2013) daher vor, der Zielformulierung gedanklich das Präfix „Bei Abschluss des Lernprozesses wird der erfolgreiche Student in der Lage sein, ...“ voranzustellen.

Lernziele beschreiben das Handlungspotenzial, das Studierende durch die erfolgreiche Teilnahme an einer Lehrveranstaltung entwickeln (sollen). Dieses Handlungspotenzial wird durch Verben ausgedrückt (Biggs u. Tang, 2011, Kap. 7), die unterschiedliche Expertisegrade differenzieren. Üblicherweise orientieren sich die Definitionen kompetenzorientierter Lernziele dabei an etablierten Lernzieltaxonomien aus der Literatur. Weit verbreitet ist die sechsstufige Taxonomie von Bloom (Bloom u. a., 1956), die mittlerweile häufig in der Überarbeitung von Anderson und Krathwohl verwendet wird (Anderson u. a., 2001). Diese unterscheidet die kognitiven Ebenen *Erinnern*, *Verstehen*, *Anwenden*, *Analysieren*, *Evaluieren* und *Kreieren*. Neben diesen sechsstufigen Taxonomien sind auch dreistufige Modelle in Gebrauch. Beispielsweise differenzieren (Schneider, 2002; Hauer, 2011) die drei Ebenen *Reproduktion*, *Anwendung* und *Übertragung*, was (Metzger u. Nüesch, 2004; UZH, 2008) zu *Wiedergeben*, *Wissen und Anwenden* sowie *Probleme bearbeiten* ausgestalten. Um die Überprüfbarkeit des Erreichens der Lernziele zu gewährleisten, wird empfohlen, ausschließlich solche Verben zu verwenden, die eine beobachtbare Handlung beschreiben (Kennedy, 2008; Hollender, 2010).

Viele Hochschulen stellen Leitfäden für die Erstellung von Modulbeschreibungen bereit. Diese legen in der Regel fest, welches Stufenmodell den Lernzieldefinitionen zugrunde liegen soll und welche Verben jeweils für die Formulierung von Lernzielen auf den verschiedenen kognitiven Taxonomiestufen genutzt werden sollen (FH Furtwangen, 2013; Uni Würzburg, 2013; Cursio u. Jahn, 2015; TUM, 2016; Bischoff u. a., 2017). Teilweise enthalten diese Leitfäden auch detailliert begründete Negativbeispiele. Einige dieser Leitfäden wirken ebenso fundiert wie pragmatisch auf den Punkt gebracht, bieten also den Modulverantwortlichen eine sehr gute Ausgangsbasis. Dennoch ist die Existenz eines hervorragenden Leitfadens zur Erstellung von kompetenzorientierten Modulbeschreibungen an einer Institution kein Garant dafür, dass die an dieser Institution existierenden Modulbeschreibungen auch tatsächlich alle diese wohl durchdachten Vorgaben erfüllen.

Akkreditierungsagenturen verlangen in ihren Richtlinien üblicherweise die Beschreibung von Inhalten und Qualifikationszielen, ohne dabei jedoch nähere Vorgaben zu deren Formulierung bzw. Detailgrad festzulegen, siehe z. B. (ACQUIN, 2014). Ebenfalls gefordert wird ein Nachweis für die kontinuierliche Überprüfung und Weiterentwicklung der Qualifikationsziele. Des Weiteren hinterfragen die Akkreditierungsagenturen, inwieweit das jeweilige Curriculum dazu geeignet erscheint, die definierten Qualifikationsziele auch tatsächlich zu erreichen.

Einzelne Fragmente in den Handreichungen der Akkreditierungsagenturen weisen darauf hin, dass auch hier nach wie vor eine gewisse Dozierenden-zentrierte Sichtweise besteht, die impliziert, dass Kompetenzen von den Lehrenden *vermittelt* werden können und nicht von den Lernenden selbst *entwickelt* werden müssen. Typische Negativbeispiele sind Formulierungen wie „Welche Qualifikationsziele werden vermittelt?“ (ACQUIN, 2014) bzw. „Die Studierenden werden in die Lage versetzt...“ (AQAS, 2016), welche den Eindruck erwecken, als könnten Studierende quasi *von außen* befähigt werden.

3 Ziel

Wie bereits dargelegt betrachten wir Modulbeschreibungen mit kompetenzorientierten Lernzieldefinitionen als eine wesentliche Informationsquelle, die insbesondere für Studierende und Lehrende, aber auch für Arbeitgeberinnen und Arbeitgeber sowie für Akkreditierungsagenturen von zentraler Bedeutung sind. Entsprechend sehen wir speziell die Lernzieldefinitionen als zentrales Instrument für die Gestaltung unsere Lehre sowie für die systematische Reflexion über diese. Wir wollen also weg vom rein pflichterfüllenden, schematischen Definieren und statt dessen hin zu sinnvollem, Klarheit stiftenden Inhalt.

Unser initiales Ansinnen war, ein Softwarewerkzeug zur automatisierten Qualitätsprüfung von Modulbeschreibungen zu realisieren, das bei der Erstellung bzw. Weiterentwicklung von Modulbeschreibungen deren Qualitätssicherung unterstützt, analog zu einer Unit Test Stage. Nach ersten Experimenten mit Systemen zur Sprachanalyse oder Ähnlichem wurde schnell sichtbar, dass vor der automatisierten Qualitätsprüfung zunächst eine Fülle noch offener Fragen auf konzeptueller Ebene zu klären ist, um Qualitätskriterien einerseits und typische Fehlermuster andererseits zu konkretisieren.

Dafür sind zunächst die Personengruppen zu identifizieren, welche die Lernzieldefinitionen primär nutzen, und deren spezifische Anforderungen an die Lernzieldefinitionen zu klären. Aus diesen Anforderungen ergeben sich Qualitätskriterien, die eine konkrete Lernzieldefinition erfüllen muss, damit sie diesen Anforderungen gerecht wird. Darauf aufbauend lassen sich Indikatoren entwickeln, die darauf hinweisen, ob und ggf. inwieweit diese Qualitätskriterien erfüllt oder verletzt sind. D. h. es ist zu klären, was genau eine Lernzieldefinition *good*, *bad* oder *ugly* macht.

Die dabei gewonnenen Erkenntnisse zu Qualitätskriterien und -indikatoren für kompetenzorientierte Lernzieldefinitionen teilen wir in dieser Arbeit, um

sie für andere nutzbar zu machen – auch jenseits der Automatisierung. Das Verständnis dieser Aspekte ist die Grundlage für eine systematische Qualitätssicherung von Modulbeschreibungen. Diese ist aufgrund der hohen Stückzahl von Modulen in einem Studiengang typischerweise ein sehr aufwändiges Unterfangen – aber dennoch erforderlich, nicht nur kurz vor der (Re-)Akkreditierung. Je klarer die Vorstellung darüber ist, was eine *gute* Lernzieldefinition ausmacht, umso schneller und treffsicherer geht die Qualitätssicherung von der Hand. Mit dieser Arbeit leisten wir somit einen Beitrag zur Verbesserung der Qualität von kompetenzorientierten Lernzieldefinitionen.

Fernziel wäre dann die darauf basierende Entwicklung von Templates mit Best Practices und die Ausarbeitung von Leitfäden zur Definition kompetenzorientierter Lernziele, was jedoch den Rahmen dieser Arbeit sprengen würde.

4 Anforderungen an kompetenzorientierte Lernzielbeschreibungen

Die in den Modulbeschreibungen dokumentierten Lernzieldefinitionen sind für verschiedene Personengruppen von Bedeutung:

- Aus Sicht der *Studierenden* definiert die Modulbeschreibung die Prüfungsanforderungen und bietet damit eine wichtige Orientierungshilfe für die Gestaltung der studentischen Selbstlernzeit. Im Hinblick auf eine Mobilitätsentscheidung hilft sie bei der Auswahl von äquivalenten und damit anerkennungsfähigen Angeboten im Ausland. Und nicht zuletzt vermittelt eine handlungsorientierte Beschreibung der Lernziele einen Einblick in das zu erwartende Tätigkeitsfeld des angestrebten Berufsbildes und dient damit als Entscheidungsgrundlage für eine fundierte Auswahl des „richtigen“ Studiengangs.

- Für die *Lehrenden* ist die kompetenzorientierte Lernzieldefinition elementares Handwerkszeug bei der Ausgestaltung einer Veranstaltung, inklusive des Leistungsnachweises, im Sinne des Constructive Alignment. Darüber hinaus bietet der Erstellungsprozess einer kompetenzorientierten Lernzieldefinition umfassendes Potenzial zur Selbstreflexion: Was lehre ich hier eigentlich? Und warum? Wofür ist das später wichtig?

  Auch Lehrenden, die neu in ein Modul einsteigen, dient eine kompetenzorientierte Lernzieldefinition als Orientierung bei der eigenen Ausgestaltung der Veranstaltung und der zugehörigen Prüfungen. Vorsitzende von Prüfungskommissionen wiederum nutzen die Lernzieldefinition als Entscheidungsgrundlage für die Anerkennung von Vorerfahrungen bzw. andernorts erbrachten Studienleistungen.

  Ebenfalls hochrelevant sind kompetenzorientierte Lernzieldefinitionen, um bei der Gestaltung eines Curriculums die Stringenz des Aufbaus eines Studienganges sicherzustellen. Dabei muss insbesondere eine aus Sicht des iterativ-inkrementellen Kompetenzerwerbs sinnhafte Reihenfolge der Module gewährleistet sein, welche einen sukzessiven Kompetenzzuwachs choreographiert und gleichzeitig Lücken und Redundanzen vermeidet.

- *Arbeitgeberinnen und Arbeitgeber* nutzen die kompetenzorientierten Lernzieldefinitionen zunehmend, um diese mit dem von ihnen benötigten Kompetenzprofil abzugleichen. Angesichts der Fülle der heute verfügbaren Studienangebote einerseits und der Diversität von Aufgabenstellungen auf dem Arbeitsmarkt andererseits ist gerade in der IT-Branche die Suche nach einem „Informatiker (m/w/d)“ zunehmend unüblich. Statt dessen geht der Trend dahin, Kompetenzprofile von Studienverläufen mit Anforderungsprofilen von Job-Beschreibungen abzugleichen und so den Auswahlprozess explizit fähigkeitsorientiert zu gestalten.

- Nicht zuletzt blicken die *Akkreditierungsagenturen* im Rahmen der Qualitätssicherung von Studiengängen ebenfalls auf die Lernzieldefinitionen. Sie evaluieren auf dieser Basis sowohl die Stringenz des Aufbaus des Studienganges als auch die Bedarfsgerechtigkeit der damit zu erlangenden Kompetenzen mit Blick auf die Erfordernisse von Gesellschaft und Arbeitsmarkt.

  Gleichzeitig stellen sie sicher, dass die oben genannten Bedürfnisse der jeweiligen Interessengruppen durch die verfügbaren Lernzieldefinitionen auf angemessene Weise erfüllt werden.

Jede dieser Personengruppen blickt aus ihrer individuellen Perspektive auf die Lernzieldefinitionen. Daraus ergeben sich entsprechend unterschiedliche Ausprägungen von Anforderungen.

Gemäß der heute immer noch weit verbreitet genutzten Definition von (Mager, 1972, S. 19) ist ein Lernziel „eine zweckmäßige Zielbeschreibung (...), mit der es gelingt, die Unterrichtsabsichten dem Leser mitzuteilen. Eine gute Zielbeschreibung schließt darüber hinaus eine möglichst große Anzahl möglicher Missdeutungen aus“.

Aus dieser Definition heraus ergeben sich unmittelbare Hinweise auf Qualitätskriterien. Zunächst soll die Zielbeschreibung „zweckmäßig“ sein, also einerseits die benötigten Informationen bereit stellen, ohne andererseits unnötigen Aufwand beim Erstellen bzw. Verarbeiten zu generieren. Des Weiteren teilt die Lernzieldefinition das *Ziel* bzw. die *Unterrichtsabsicht* mit. Sie fokussiert also das zu erreichende Ergebnis und nicht den Weg dahin, also nicht die Schritte des Lernprozesses. Darüber hinaus *schließt* eine gute Lern-

zieldefinition *Missdeutungen aus*. Sie ist also inhaltlich eindeutig sowie verständlich formuliert.

Daraus leiten wir die folgenden Qualitätskriterien ab:

- Effizient / auf den Punkt
- Ergebnisorientiert (nicht prozessorientiert)
- Eindeutig
- Verständlich

Dabei ist zu beachten, dass Verständlichkeit individuell sehr unterschiedlich wahrgenommen wird. Insbesondere haben Studierende bzw. Studieninteressierte, die noch über wenig bis gar kein Domänenwissen verfügen, hier völlig andere Bedarfe und eine andere Wahrnehmung als Personen, die in dieser Domäne als Experten gelten.

5 Vorgehensweise

Basierend auf diesem Grundverständnis der Anforderungen der verschiedenen Personengruppen an Lernzieldefinitionen sowie der abgeleiteten Qualitätskriterien sind im nächsten Schritt Indikatoren zu identifizieren, die darauf hinweisen, ob eine konkrete Lernzieldefinition im Sinne dieser Anforderungen und Qualitätskriterien *good*, *bad* oder *ugly* ist.

5.1 Prozess

Zentraler Bestandteil einer kompetenzorientierten Lernzieldefinition ist neben der Inhaltskomponente auch das intendierte Handlungspotenzial. Dieses wird in der Regel ausgedrückt über Verben. In unserem ersten Ansatz der Analyse von Lernzieldefinitionen haben wir daher diese Verben automatisiert aus den ersten von uns gesammelten Lernzieldefinitionen extrahiert mit Hilfe des Stanford Part-of-Speech-Tagger (Toutanova u. a., 2003), welcher in der Webseite `wortarten.info` für die Deutsche Sprache integriert ist. Diese automatisierte Analyse hat zwar zu ersten empirisch begründeten Erkenntnissen geführt, aber schnell klar gemacht, dass ergänzend zu den Verben noch weitere Indikatoren zu betrachten sind.

Um diese Indikatoren zu identifizieren haben wir eine Vielzahl von Modulbeschreibungen gesammelt, deren Lernzieldefinitionen analysiert und diese dann als *good* oder *bad* im Sinne der oben genannten Qualitätskriterien klassifiziert. Dabei haben wir für konkrete Beispiele herauspräpariert, was genau an diesen Lernzieldefinitionen jeweils gut oder schlecht ist. Die so auf der konkreten Ebene identifizierten Indikatoren haben wir anschließend hochabstrahiert zu Kriterien und das jeweils zugrunde liegende Prinzip benannt, sowohl für positive als auch für negative Beispiele.

Abschließend haben wir diese abstrahierten Kriterien nach ihrer Relevanz gewichtet bzw. sortiert. Insbesondere für die Negativ-Kriterien ergab sich daraus eine Abstufung zwischen *the Bad* und *the Ugly* (siehe Abbildung 1).

Bei dieser Gewichtung der abstrahierten Kriterien unterscheiden wir insbesondere, ob bei einem Verstoß gegen ein Positiv-Kriterium die Lernzieldefinition auf semantische oder auf syntaktische Weise hässlich wird. Als semantisch hässlich betrachten wir die Verletzung von Kriterien, welche die Eindeutigkeit (also die Sicherheit vor Missdeutung) fokussieren bzw. die Ergebnisorientierung der Lernzieldefinition gewährleisten. Diese Kriterien sehen wir als besonders relevant an. Wird in einer konkreten Lernzieldefinition eines oder mehrere dieser Kriterien verletzt, fällt diese Lernzieldefinition in die Äquivalenzklasse *the Bad*.

Im Gegensatz dazu klassifizieren wir die Verletzung von Kriterien, welche primär das Qualitätskriterium *Effizienz* adressieren, als syntaktisch hässlich. Syntaktisch hässliche Lernzieldefinitionen sind zwar in ihrer inhaltlichen Aussage prinzipiell klar definiert, aber so ungünstig in Sprache verpackt, dass diese inhaltliche Aussagekraft nur mit entsprechender Mühe (und in der Folge mit einer gewissen „Unfallgefahr“) erkennbar ist. Derartige Lernzieldefinitionen sind Repräsentanten von *the Ugly*.

Am Übergang zwischen *the Bad* and *the Ugly* liegen Lernzieldefinitionen, die gegen das Qualitätskriterium *Verständlichkeit* verstoßen. Wenn eine bestimmte Person eine Lernzieldefinition nicht versteht, ist das aus Sicht dieser Person ähnlich schlimm, als wenn die Lernzieldefinition semantisch uneindeutig ist – auch wenn anderen Personen die Bedeutung dieser Lernzieldefinition klar ist. Dieses Kriterium ist insbesondere mit Blick auf die Personengruppe der Studierenden bzw. Studieninteressierten von großer Bedeutung. Diese sind in der Regel zu dem Zeitpunkt, zu dem sie sich mit einer bestimmten Modulbeschreibung auseinander setzen, mit den in diesem Modul adressierten Inhalten noch nicht vertraut, da diese im Verlauf des Moduls ja erst gelernt werden sollen. Um die Verständlichkeit einer Lernzieldefinition sicherzustellen müssen bei der Erstellung die unterschiedlichen Expertise-Ebenen und fachlichen Hintergründe der verschiedenen Zielgruppen entsprechend berücksichtig werden.

5.2 Datenbasis

Als Databasis für unsere Analysen haben wir von 39 deutschen Hochschulen für angewandte Wissenschaften sowie von fünf deutschen Universitäten Modulbeschreibungen gesammelt, aus dem fachlichen Umfeld von Softwareentwicklung bzw. Software Engineering in Informatik-nahen Studiengängen. Betrachtet wurden dabei diejenigen Hochschulen für angewandte Wissenschaften bzw. Universitäten, die in den Ranglisten des Hochschulrankings der Wirtschaftswoche von 2018 sowie dem CHE-Ranking auf den oberen Plätzen aufgelistet sind, denen also eine hohe Qualität in der Lehre attestiert wurde. Ebenfalls betrachtet wurden alle Hochschulen aus dem UAS7-Verbund, die sich selbst als „dem höchsten Qualitätsstandard in [der] Lehre [...] verpflichtet“ (www.uas7.de) sehen. Ergänzend wurden die Hochschulen und Universitäten in

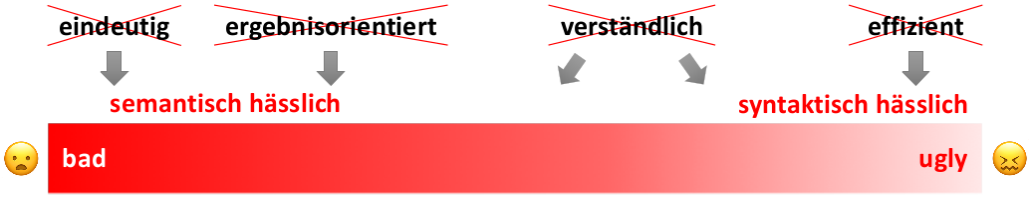


Abbildung 1: Verletzung von Qualitätskriterien und deren Folgen

unserer unmittelbaren geographischen Nachbarschaft mit betrachtet.

Im nächsten Schritt wurden an den so ausgewählten Institutionen die Informatik-nahen Studiengänge im Bachelor-Bereich identifiziert, mit Fokus auf „Informatik“, „Allgemeine Informatik“, „Angewandte Informatik“ und „Praktische Informatik“. Innerhalb dieser Studiengänge konzentrierten wir uns dann auf all diejenigen Module, die „Softwareentwicklung“ und „Software Engineering“ adressieren, also auch „(Objektorientierte) Programmierung“, „Softwaretechnik“, „Programmiermethodik“, „Techniken der Programmentwicklung“ sowie Modulen zu konkreten Programmiersprachen. Ebenfalls berücksichtigt wurden Module wie „Projekt“ bzw. „Projektarbeit“, wenn diese erkennbaren Bezug zu Softwareentwicklung bzw. Software Engineering aufwiesen.

Insgesamt wurden auf diese Weise rund 250 Modulbeschreibungen von Hochschulen für angewandte Wissenschaften sowie rund 25 Modulbeschreibungen von Universitäten ausgewählt. Für jedes dieser Module wurden die folgenden Daten gesammelt:

- *Organisatorische Rahmendaten*
  Betrachtet wurden dabei grundlegende Informationen zur organisatorischen Einbettung des Moduls, wie beispielsweise Hochschule, Studiengang, Lehrveranstaltung, Modul, Fakultät/Fachbereich, URL, ECTS, SWS, Arbeitsaufwand und Prüfungsform. Obwohl diese Daten selbst nicht direkt im Mittelpunkt unserer Untersuchung standen waren sie hilfreich, um eine grundlegende Vorstellung der jeweiligen Module zu entwickeln und sie in ihren fachlichen und organisatorischen Kontext einzuordnen.
- *Akkreditierungsdaten*
  Hier wurde erfasst, ob der Studiengang, in dem das Modul verortet ist, bereits akkreditiert wurde und falls ja, wann und von welcher Akkreditierungsagentur die Akkreditierung erteilt wurde. Diese Daten sind für unsere Untersuchung insofern relevant, als Modulbeschreibungen in der Regel im Rahmen von Akkreditierungsverfahren qualitätsgesichert werden. Des Weiteren zeigt die Selbsterfahrung, dass unterschiedliche Akkreditierungsagenturen sich in ihren Vorgaben und Präferenzen hinsichtlich kompetenzorientierter Lernzieldefinitionen in Modulbeschreibungen durchaus unterscheiden. Entsprechend vermuten wir eine Korrelation zwischen der jeweiligen Akkreditierungsagentur und der Form sowie ggf. Qualität der Lernzieldefinitionen in den Modulbeschreibungen.
- *Lernzieldefinitionen, Inhalte und Lehrformen*
  Diese Daten lagen im eigentlichen Fokus unserer Untersuchung. Um bei unserer Analyse eine entsprechende Präzision zu gewährleisten und Missdeutungen zu vermeiden wurde explizit dokumentiert, wie die einzelnen Institutionen innerhalb ihrer Modulbeschreibungen die jeweiligen Rubriken für Lehr-Lernziele, Inhalte und Lehrform genau bezeichnen und inwieweit diese Rubriken syntaktisch voneinander abgegrenzt sind.

6 The Good

Abbildung 2 vermittelt eine sortierte Übersicht über die Kriterien für gute bzw. schlechte Lernzieldefinitionen, also für *the Good, the Bad and the Ugly*. Zunächst betrachten wir die positiven Beispiele und damit diejenigen Kriterien, deren Erfüllung eine Lernzieldefinition als eine Instanz von *the Good* qualifizieren.

Inhalt und Handlungskomponente

Jede gute Lernzieldefinition kombiniert einen Inhalt mit einer Handlungskomponente. Diese beschreibt, welche Art von Tätigkeit auf bzw. mit diesem Inhalt durchzuführen ist. In der Regel wird diese Handlungskomponente über ein Verb ausgedrückt.

Ein Beispiel für eine Lernzieldefinition, die Inhalt und Handlungskomponente kombiniert, wäre „Die Studierenden (...) können einfache Funktionalitäten in Klassen kapseln, Objekte erzeugen und Methoden aufrufen. Darüber hinaus sind sie in der Lage, korrekten, lesbaren und wartbaren Code zu erzeugen“ (HS Fulda, 2018).

Studierendenzentrierung

Lernziele beschreiben das Handlungspotenzial, das Studierende durch die erfolgreiche Teilnahme an einer Lehrveranstaltung entwickeln (sollen). Die mit diesem Handlungspotenzial verbundenen Kompetenzen werden dabei nicht von außen in die Studierenden „eingefüllt“, sondern aktiv von den Studierenden erworben. Die in der Lernzieldefinition verwendeten Verben entsprechen dieser Sichtweise, beschreiben also das von den Studierenden zu erlernende Ver-

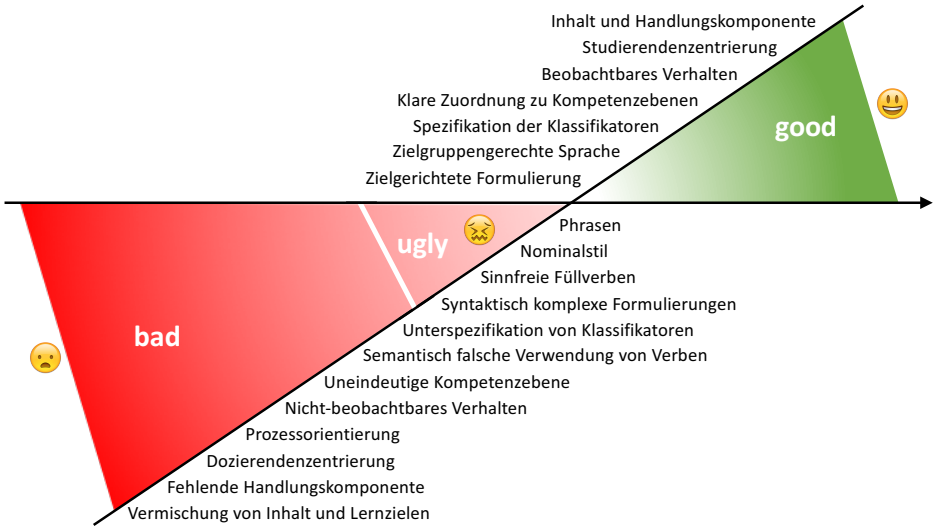


Abbildung 2: Kriterien für die Bewertung von Lernzieldefinitionen – von *bad* über *ugly* nach *good*

halten, wie im obigen Beispiel der HS Fulda. (Eine Dozierenden-zentrierte Formulierung würde dagegen beschreiben, was die Lehrperson „vermitteln“ will, also das Lehrziel der Lehrperson und nicht das Lernziel der Studierenden.)

Ergebnisorientierung

Ein Lernziel definiert, welche Fähigkeiten die Studierenden im Laufe des Lernprozesses entwickelt haben sollen, also die als Ergebnis gewünschten Kompetenzen, wie im obigen Beispiel der HS Fulda. Sie beschreiben dagegen *nicht* die einzelnen Schritte, die im Zuge des Lernprozesses zu durchlaufen sind, um diese Kompetenzen aufzubauen.

Beobachtbares Verhalten

Gemäß der aktuell in Deutschland gängigen Lehr-Lernpraxis muss spätestens am Ende einer Modul-Durchführung überprüft werden, inwieweit die einzelnen Studierenden die Lernziele erreicht haben. Damit klar wird, wann ein Lernziel als erreicht gilt, ist die Handlungskomponente in Form von beobachtbarem Verhalten zu beschreiben.

Die Lernzieldefinition „Nach erfolgreicher Beendigung der Veranstaltung sind die Studierenden in der Lage, verschiedene Vorgehensmodelle mit ihren Stärken und Schwächen zu beschreiben“ (HS Kempten, 2018) der Hochschule Kempten definiert, welches beobachtbare Verhalten die Studierenden zeigen müssen, damit das Lernziel als erreicht gilt. Dadurch wird das Erreichen des Lernziels überprüfbar.

Klare Zuordnung zu Kompetenzebenen

Die Handlungskomponente eines Lernziels legt die Art der Tätigkeit fest, die auf bzw. mit dem Inhalt durchzuführen ist. Sie repräsentiert also das Kompetenzniveau, d. h. den zu erreichenden Grad an handlungsbefähigender Expertise. Guten Lernzieldefinitionen liegt ein wohldefinierte Taxonomie von Kompetenzebenen zugrunde, wie beispielsweise (Bloom u. a., 1956), (Anderson u. a., 2001), (Schneider, 2002), (Metzger u. Nüesch, 2004) oder (Hauer, 2011).

Einige Hochschulen legen in ihren Leitfäden zur Formulierung von Modulbeschreibungen fest, welche Taxonomie für Kompetenzebenen verwendet werden soll. Teilweise werden auch die zu nutzenden Verben vorgegeben. Zu beachten ist, dass die gleichen Verben bei verschiedenen Taxonomien mit unterschiedlichen Kompetenzebenen korrespondieren. Beispielsweise liegt *Beurteilen* in der sechsstufigen Taxonomie von (Bloom u. a., 1956) auf Ebene *6: Evaluieren*, während in der ebenfalls sechsstufigen Überarbeitung von (Anderson u. a., 2001) *Evaluieren* auf Ebene 5 liegt. Im dreistufigen Modell von (Metzger u. Nüesch, 2004) wiederum korrespondiert *Beurteilen* mit der dritten Ebene *Probleme bearbeiten*.

Der erste Satz der nachfolgenden Lernzieldefinition „Die Studierenden kennen weitergehende Konzepte der nebenläufigen und verteilten Programmierung. Sie können beurteilen, wann und wie ein Algorithmus sich erfolgreich parallelisieren lässt (...)“ (FH Münster, 2018) adressiert beispielsweise die Kompetenzebene *1: Erinnern* nach der Lernzieltaxonomie von (Anderson u. a., 2001), während der zweite Satz auf der Kompetenzebene *5: Evaluieren* angesiedelt ist.

Spezifikation der Klassifikatoren

Insbesondere auf den höheren Kompetenzebenen im Sinne der oben genannten Taxonomien finden sich immer wieder Formulierungen wie „an einfachen Aufgabenstellungen bzw. Beispielen“. Die Erfahrung zeigt, dass es hochgradig von der bereits vorhandenen Expertise abhängt, ob eine Person eine Aufgabenstellung bzw. ein Beispiel als „einfach“ empfindet oder nicht.

Tabelle 1: Korrespondierende Paare von Kriterien für *good* vs. *bad / ugly* und deren Relevanz

Relevanz	**The Good**	**The Ugly and the Bad**	Relevanz
++++	Inhalt und Handlungskomponente	Vermischung von Inhalt und Lernzielen	– – – –
		Fehlende Handlungskomponente	– – – –
+++	Studierendenzentrierung	Dozierendenzentrierung	– – –
+++	Beobachtbares Verhalten	Prozessorientierung	– – –
		Nicht-beobachtbares Verhalten	– – –
++	Klare Zuordnung zu Kompetenzebenen	Uneindeutige Kompetenzebene	– –
		Semantisch falsche Verwendung von Verben	– –
++	Spezifikation der Klassifikatoren	Unterspezifikation von Klassifikatoren	– –
++	Zielgruppengerechte Sprache	Syntaktisch komplexe Formulierungen	– –
+	Zielgerichtete Formulierung	Sinnfreie Füllverben	–
		Nominalstil	–
		Phrasen	–

Um die Lernzieldefinition eindeutig zu gestalten und Missdeutungen möglichst auszuschließen ist die Bedeutung derartiger Klassifikatoren klar zu beschreiben.

Ein Beispiel für eine Lernzieldefinition mit klar spezifiziertem Klassifikator wäre: „Die Studierenden entwickeln für ein einfaches Problem aus einer gegebenen Anforderungsspezifikation heraus einen Entwurf, der sowohl die Gesamtstruktur der Lösung als auch die einzelnen Algorithmen vorgibt. (...) Ein „einfaches Problem“ ist dabei eine Aufgabenstellung, die mit max. 10 Klassen objektorientiert zu lösen ist“ (HS München, 2018).

Zielgruppengerechte Sprache

Damit die Zielgruppe eine Lernzieldefinition korrekt interpretieren kann ist es wichtig, diese in eine Sprache zu kleiden, die für die Zielgruppe auch verständlich ist. Mit Blick auf die Studierenden ist hier besonders wichtig, dass Fachbegriffe nur moderat verwendet und deren Bedeutung ggf. knapp erläutert wird. Ebenfalls hilfreich ist es, Sätze kurz zu halten und möglichst einfache Formulierungen zu verwenden.

Zielgerichtete Formulierung

Eine Lernzieldefinition empfinden wir dann als zielgerichtet, wenn die erforderlichen Informationen so ausführlich wie nötig, aber so knapp wie möglich dargestellt werden. Die erforderliche Information muss also vollständig und angemessen detailliert sein, ohne dabei unnötig breit getreten oder mit Phrasen aufgebauscht zu werden. Zu beachten ist, dass das Empfinden, welcher Umfang bzw. Detaillierungsgrad angemessen ist, ein Stück weit Geschmackssache ist.

7 What makes the Bad bad and the Ugly ugly?

Nach diesen positiven Beispielen widmen wir uns im Folgenden der Analyse von negativen Beispielen. Wir untersuchen also, welche Kriterien dazu führen, dass eine Lernzieldefinition semantisch oder syntaktisch hässlich, also *bad* oder *ugly* ist, und an welchen Indikatoren das schnell und klar erkennbar ist. Wir stellen dabei eine Menge von Mustern (Anti-Patterns) zusammen, die wir aus den Formulierungen der untersuchten Modulbeschreibungen heraus abstrahiert haben und die wir als ungünstig für die Definition kompetenzorientierter Lernziele betrachten. Diese Formulierungen müssen nicht per se immer negativ sein. Es lohnt sich jedoch, in jedem Einzelfall nachzudenken, ob ihre Verwendung für den Zweck der kompetenzorientierten Lernzieldefinition zielführend ist – im Sinne des Kapitels „Giftschrank der Wörter“ aus (Rechenberg, 2006).

Anders als bei den positiven Beispielen aus Abschnitt 6 geben wir zu den Negativbeispielen keine Quellen an, denn wir wollen mit unserer Arbeit kein Fingerpointing schüren, sondern zu Reflexion anregen. Aus dem gleichen Grund haben wir darauf geachtet, als Beispiele jeweils so kurze Fragmente aus den zugrunde liegenden Lernzieldefinitionen herauszuschneiden, dass diese nicht mehr eindeutig durch eine einfache Internet-Suche zu identifizieren sind. In seltenen Fällen haben wir den vorgefundenen Wortlaut auch marginal abgewandelt, damit eine Suche ins Leere läuft.

Zu beachten ist ferner, dass die Zuordnung einer Formulierung zu einem Kriterium nicht immer eindeutig ist; d. h., manche Formulierungen qualifizieren sich für mehrere Arten von Hässlichkeit. Ergänzend zur sortierten Übersicht über die Kriterien für gute bzw. schlechte Lernzieldefinitionen in Abbildung 2 stellt Tabelle 1 diese Kriterien einander gegenüber, als jeweils korrespondierende Paare für die positive bzw. negative Ausprägung des gleichen Anliegens.

7.1 Semantisch hässlich – the Bad

Wir beginnen die Diskussion des Verbesserungspotenzials für Lernzieldefinitionen mit denjenigen Kriterien, die besonders gravierend sind, d. h. in besonderem Maße dazu beitragen, dass eine Lernzieldefinition zu *the Bad* zählt.

Vermischung von Inhalt und Lernzielen

Ein kompetenzorientiertes Lernziel kombiniert jeweils einen Inhalt mit einer Handlungskomponente. Es drückt also ein Handlungspotenzial aus, das die lernende Person nach erfolgreichem Lernprozess auf bzw. mit dem Inhalt durchführen können soll.

Die Struktur der meisten untersuchten Modulbeschreibungen weist definierte, voneinander getrennte Rubriken für Inhalte und für Lernziele aus. Ideal wäre es, wenn die Rubrik „Inhalt“ einen schnellen Überblick über die im Modul adressierten Themengebiete vermitteln würde, während die angestrebten Kompetenzen (als Kombination von Inhalt und Handlungskomponente) in der Rubrik „Lernziele“ angegeben werden.

Zu beobachten ist jedoch, dass diverse Modulbeschreibungen diese Rubriken nicht trennscharf nutzen, sondern vielmehr reine Inhalte in die Rubrik für Lernziele bzw. Kompetenzen schreiben. Umgekehrt werden gelegentlich auch die zu entwickelnden Kompetenzen in der Rubrik für die Inhalte dokumentiert.

Durch diese strukturelle Unsauberkeit werden die Modulbeschreibungen vergleichsweise schlecht lesbar. Insbesondere wird es für die Zielgruppen der Modulbeschreibung ungleich schwieriger, zweifelsfrei die in diesem Modul adressierten Kompetenzen zu identifizieren – insbesondere dann, wenn die Lernziele nicht in der Rubrik „Lernziele“ aufgelistet sind, sondern versteckt unter einer anderen Überschrift. Aber auch die Einbettung einzelner Lernziele zwischen umfangreichen inhaltlichen Erläuterungen macht die Lernziele schwer erkennbar, selbst wenn sie letztlich doch in der passenden Rubrik aufgeführt sind. Des Weiteren liegt der Verdacht nahe, dass die Urheber derartiger Modulbeschreibungen nicht angemessen präzise für sich selbst geklärt haben, wie nun eigentlich Inhalte und Lernziele zueinander in Beziehung stehen und welche Kompetenzen die Studierenden überhaupt in diesem Modul entwickeln sollen.

Beispiele für derartige, zwischen die Lernziele eingestreute inhaltliche Erläuterungen sind:

- „Softwareentwicklung wird als iterativer und inkrementeller Prozess (...) verstanden“
- „Die Entwicklung von Softwaresystemen (...) [ist] in der Praxis immer fächerübergreifend“
- „Software Engineering umfasst (...)“

Einige dieser ausschließlich inhaltlichen Erläuterungen muten an wie eine motivatorische Begründung der Relevanz der behandelten Inhalte. Diese ist ja durchaus eine relevante Information, sollte aber an einer dafür geeigneteren Stelle stehen, beispielsweise in der Studiengangsbeschreibung oder eben in der Rubrik mit den Inhalten.

Des Weiteren finden sich nach wie vor diverse Modulbeschreibungen, die zwar Inhalte auflisten, aber keinerlei Lernziele definieren.

Fehlende Handlungskomponente

Jedes kompetenzorientierte Lernziel beinhaltet eine Handlungskomponente. Diese wird über ein Verb ausgedrückt, wie beispielsweise „beurteilen“ oder „erstellen“ – oder zumindest die Nominalisierung von Verben, wie beispielsweise „Kenntnis“ oder „Anwendung“.

Zu beobachten ist, dass einige Lernzieldefinitionen keine kompetenzbeschreibende Handlungskomponente beinhalten, weder als Verb noch als Nominalisierung eines Verbs. Statt dessen werden entweder reine Inhalte aufgelistet oder aber Hilfsverben verwendet, die lediglich den Lernprozess an sich beschreiben, jedoch nicht das angestrebte Handlungspotenzial ausdrücken.

Dadurch wird nicht deutlich, welche Form von Handlungspotenzial auf und mit dem jeweiligen Inhalt als Lernergebnis erwartet wird. Die zu erreichende Kompetenz bleibt damit völlig unklar – sind lediglich Definitionen wiederzugeben oder ist ein fachliches Artefakt zu entwickeln, das den etablierten Qualitätskriterien der Zunft genügt?

Beispiele für Lernzieldefinitionen, in welchen die angestrebte Handlungsebene komplett offen bleibt, sind:

- „(...) lernen die Studierenden weitergehende Methoden ... [gefolgt von einer Auflistung von Technologien]“
- „Daneben erwerben sie Kompetenz in ... [gefolgt von einer Aufzählung reiner Inhalte]“
- „Steigerung der (...) Fähigkeiten in (...)“

Dozierendenzentrierung

Dozierendenzentrierte Formulierungen beschreiben, was die Dozierenden zu lehren gedenken, häufig mit einem starken Fokus auf reine Inhalte. Das Handlungspotenzial und damit die Kompetenz, die die Studierenden durch Teilnahme an der Lehr-Lernveranstaltung entwickeln können, wird dagegen nicht definiert. Entsprechend beschreiben derartige Formulierungen Lehrziele und nicht Lernziele.

Letztere bleiben dabei völlig unklar. Insbesondere wird das von den Studierenden erwartete Handlungspotenzial und damit die Kompetenzebene bei diesen Formulierungen nicht deutlich. Des Weiteren weisen dozierendenzentrierte Formulierungen den Studierenden in ihrem Lernprozess eine passive Rolle zu, die einer modernen Lehr-Lernsituation nicht angemessen ist. Derartige Modulbeschreibungen lesen sich dann eher wie Drohungen, nicht wie Versprechen.

Beispielsweise behandeln die folgenden Formulierungen die Studierenden wie passive Objekte des Lehrprozesses:

- „die Studierenden werden in die Lage versetzt...“ (AQAS, 2016)
- „die Studierenden werden befähigt zu ...“
- „Weiterhin erfahren [die Studierenden] eine Einführung in ...“

Im Extremfall kommt eine Lehrzielbeschreibung ganz ohne Bezug zu Studierenden aus:

- „Die Lehrveranstaltung führt ein in ...".
- „Das Modul vermittelt ..."
- „Außerdem wird ... behandelt"
- „In der Veranstaltung wird ... vorgestellt"
- „Die Programmierkenntnisse in Java werden erweitert um ..."

Typische Indikatoren für dozierendenzentrierte Formulierungen sind also Verben wie „vermitteln", „behandeln" oder „befähigen", sowie deren Nominalisierung, also beispielsweise „die Vermittlung von".

Prozessorientierung

Einige Modulbeschreibungen dokumentieren als Lernziele weniger die gewünschten Lernergebnisse als den Prozess bzw. die Lernsituation, der verfolgt bzw. die aufgebaut wird, um diese Lernergebnisse zu erreichen.

Derartige Formulierungen sind aus zweierlei Gründen nicht ideal. Zum einen bleibt die zu erlernende Handlungskomponente in der Regel weit gehend unklar. Zum anderen ist es prinzipiell denkbar, dass es verschiedene mögliche Wege zum gleichen Ziel gibt. Eine prozessorientierte Modulbeschreibung schränkt damit unnötig ein auf den einen vorgegebenen Lernpfad.

Typische Indikatoren für prozessorientierte Formulierungen sind Wörter, die eine zeitliche Reihenfolge einzelner Schritte bzw. eine Abfolge von Themen implizieren, ohne dabei die Handlungskomponente genauer auszuweisen, wie beispielsweise „ausgehend von".

- „Ausgehend von [Artefakt] steht dabei die Phase ... im Mittelpunkt."
- „Einsatz [bestimmter Verfahren] ausgehend vom [Artefakt]"

Ebenfalls häufig vertreten sind listenartige Aufzählungen der durchzuführenden Arbeitsschritte (meist als Nominalisierung) bzw. der zu verwendenden Werkzeuge, sowie eine zwischen die Lernziele gemischte Beschreibung der Lernsituation.

- „Nutzung von [Verfahren]"
- „Einsatz von [Werkzeug]"
- „... werden in Gruppen von [n] Studierenden bearbeitet ..."

Nicht-beobachtbares Verhalten

Nach den Konventionen unseres aktuellen Bildungssystemes ist im Verlaufe eines Bildungsprozesses immer wieder auf geeignete Weise zu überprüfen, inwieweit die Lernenden die gesetzten Lernziele auch erreicht, also die geforderten Kompetenzen entwickelt haben. Ob eine Person über bestimmte Fähigkeiten und Fertigkeiten verfügt kann von außen betrachtet ausschließlich am beobachtbaren Verhalten dieser Person fest gemacht werden, oder an den beobachtbaren Ergebnissen dieses Verhaltens. Geprüft wird also, ob eine Person bei bestimmten Aufgaben ein den Aufgaben angemessenes Verhalten zeigt.

Um die Erwartungshaltung an die Lernenden klar zu definieren ist es daher sinnvoll, Lernziele so zu formulieren, dass sie auf möglichst differenzierte Weise ein von außen beobachtbares Verhalten bzw. Ergebnis beschreiben. Von diesem aus kann dann auf die Existenz der geforderten Kompetenz geschlossen werden.

Nicht alle Verben beschreiben ein Verhalten, das tatsächlich beobachtbar ist. Insbesondere trifft dies auf die Bezeichner der kognitiven Ebenen der Taxonomie von Anderson und Krathwohl (Anderson u. a., 2001) zu: *Erinnern*, *Verstehen*, *Anwenden*, *Analysieren*, *Evaluieren* und *Kreieren* sind für sich genommen keine von außen wahrnehmbaren Verhaltensweisen.

Um beispielsweise beobachten zu können, ob Studierende einen Sachverhalt verstanden haben, ist eine konkrete Aufgabenbeschreibung erforderlich, an deren Bearbeitungsprozess bzw. erzieltem Ergebnis sich der Grad des Verständnisses ablesen lässt. Geeignete Formulierungen bzw. Verben für ein Verhalten, an dem *Verstehen* beobachtet werden kann, sind beispielsweise „... mit eigenen Worten beschreiben" oder „veranschaulichen".

Uneindeutige Kompetenzebene

Nicht alle Verben spezifizieren die angestrebte Expertise-Ebene so genau, dass zweifelsfrei klar ist was die Lernenden nach dem Lernprozess wirklich leisten können sollen. Auch wenn Lernzieldefinitionen basierend auf einer der bekannten Lernzieltaxonomien formuliert werden, lassen sich nicht alle denkbaren Verben eindeutig einer kognitiven Ebene zuordnen.

Dies könnte mit ein Grund dafür sein, warum sehr häufig die Ebenenbezeichner selbst verwendet werden, um Lernziele zu beschreiben – denn diese Ebenenbezeichner sind ja per Definitionem den Ebenen eindeutig zugeordnet. Sie beschreiben jedoch, wie oben dargestellt, kein beobachtbares Verhalten. Auf diese Weise definierte Lernziele sind somit nicht überprüfbar.

Typische Formulierungen, bei denen die von einem Lernziel adressierte Expertiseebene der Handlungskomponente völlig offen bleibt, sind:

- „Fähigkeit zur [Paradigma-Adjektiv] Programmierung"
- „beherrschen"
- „sinnvoll einsetzen"
- „lernen [Konzept oder Verfahren] kennen"

Semantisch falsche Verwendung von Verben

Einige der analysierten Modulbeschreibungen beinhalten Lernzieldefinitionen, die Verben auf eine Weise verwenden, dass diese Verben auf den ersten Blick semantische Assoziationen wecken mit einer bestimmten Kompetenzebene, jedoch eigentlich einer ganz anderen Kompetenzebene zugehören.

Dadurch wird die Lernzieldefinition schwer lesbar. Es steigt das Risiko von Missdeutungen.

Zu Formulierungen, die entsprechend risikobehaftet sind, zählen beispielsweise

- „verstehen [etwas] sinnvoll einzusetzen"

- „erkennen ... die Anwendbarkeit [von Konzepten für eine Problemstellung]“

Das erste Beispiel wirkt auf den ersten Blick so, als würde Ebene 2, *Verstehen*, der Taxonomien von Bloom (Bloom u. a., 1956) oder Anderson und Krathwohl (Anderson u. a., 2001) adressiert. De facto geht es bei genauerem Hinsehen jedoch darum, etwas „einzusetzen“ und damit mindestens um Ebene 3, *Anwenden*. Im zweiten Beispiel fallen zunächst die Begriffe „erkennen“ und „Anwendbarkeit“ ins Auge, die auf die Ebenen 1, *Erinnern* oder 3, *Anwenden* hinweisen. De facto geht es jedoch darum, zu identifizieren, welches der gelernten Konzepte am besten für die Lösung der Problemstellung geeignet ist, und dieses dann auszuwählen. Diese Kompetenz entspräche jedoch *Evaluieren*, also Ebene 5 der Taxonomie von Anderson und Krathwohl, bzw. Ebene 6 in der Taxonomie von Bloom.

Unterspezifikation von Klassifikatoren für Inhalt

Einige Lernzieldefinitionen klassifizieren die Inhaltskomponente, meist durch Verwendung von Adjektiven, die den Inhalt genauer beschreiben. Diese Klassifikatoren sind jedoch nur dann wirklich hilfreich, wenn ihre Bedeutung ohne Interpretationsspielraum festgelegt ist. Beispielsweise hängt es von der bereits verfügbaren Expertise ab, ob eine Person einen Algorithmus oder ein Problem als „einfach“ empfindet oder nicht. Meist wird die Bedeutung derartiger Klassifikatoren jedoch offen gelassen.

Typische Beispiele sind:

- „grundlegende Konzepte“
- „kleinere Problemstellungen“
- „eine einfache Anwendung“

7.2 Syntaktisch hässlich – the Ugly

Die bisher betrachteten Kriterien weisen auf semantische Schwierigkeiten hin, insbesondere auf Unklarheiten bzw. Unterspezifikation und damit auf hohen Interpretationsspielraum. Sie bergen somit ein Risiko auf Missdeutungen.

Im Folgenden diskutieren wir syntaktische Kriterien, die dazu führen, dass Lernzielformulierungen schwer lesbar und damit *ugly* werden.

Syntaktisch komplexe Formulierungen

Lernzieldefinitionen sollen insbesondere auch für Studierende oder Studieninteressierte verständlich sein, also eine Personengruppe, welche mit den in den Lernzielen thematisierten Inhalten zunächst nicht oder nur wenig vertraut ist. Entsprechend ist darauf zu achten, eine möglichst einfache, klar verständliche Sprache zu wählen. Diese wird nicht nur den Studierenden besser gerecht, sondern ist ggf. auch für die anderen Zielgruppen klarer und effizienter verständlich.

Beispiele für typische Fallstricke in den Formulierungen sind:

- Hoher Anteil an Fremdwörtern und Fachsprache, wenn es „normalverständlich“ auch gehen würde
- Stark verschachtelte Sätze
- Schlechte Strukturierung
- Vermischung von reinem Inhalt, Lernzieldefinitionen und Beschreibung des Lernprozesses an sich bis zur Unkenntlichkeit
- Zu geringer oder zu großer Umfang (d. h. unterspezifiziert vs. überdetailliert)

Sinnfreie Füllverben

Als *sinnfreie Füllverben* in Lernzieldefinitionen bezeichnen wir Verben, die keine Handlungskomponente mit Bezug zu Inhalt ausdrücken. Häufig treten diese sinnfreien Füllverben in Kombination mit Nominalstil auf. D. h. die eigentlich im Fokus stehende Handlungskompetenz wird in ein substantiviertes Verb verpackt. Beispiele sind:

- „verfügen die Studierenden über ein grundlegendes Verständnis“
- „beherrschen Grundkenntnisse“
- „erlernen [den] Tooleinsatz“

Nominalstil

Von Nominalstil sprechen wir, wenn die Handlungskomponente nicht über ein Verb, sondern über dessen Substantivierung ausgedrückt wird. Häufig tritt dieses Phänomen in Kombination mit sinnfreien Füllverben oder Phrasen auf. Übermäßig verwendeter Nominalstil macht einen Text (und damit auch eine Lernzieldefinition) meist schwerer verständlich.

Typische Beispiele sind, ergänzend zu den bereits bei *Sinnfreie Füllverben* genannten:

- „Vermittlung von Grundkenntnissen“
- „Umsetzung von Algorithmen“
- „Strukturierung von Daten“
- „Einsatz von Werkzeugen“

Phrasen

Ein identifiziertes Qualitätskriterium für Lernzieldefinitionen ist die *Effizienz*, also ob eine Lernzieldefinition die relevante Information auf den Punkt bringt. Kontraproduktiv dafür sind Phrasen, also wohltönende, aber inhaltlich nichtssagende Textfragmente, da sie die Lernzieldefinitionen unnötig aufblähen.

Typische Repräsentanten für Phrasen in Lernzieldefinitionen sind:

- „Die Studierenden werden befähigt ...“
- „Die Studierenden sollen ... verstanden haben“
- „Die Studierenden sind in der Lage ...“
- „Weiterhin erwerben [die Studierenden] Kenntnisse und Fertigkeiten im Bereich ...“
- „... soll ... Gelegenheit bieten ...“

8 Zusammenfassung und Ausblick

Basierend auf der Analyse der Lernzieldefinitionen aus rund 275 deutschsprachigen Modulbeschreibungen aus dem fachlichen Umfeld von Programmierung und Softwaretechnik haben wir Qualitätskriterien für Lernzieldefinitionen definiert. Auf dieser Grundlage haben wir sowohl Best Practices als auch Anti-Patterns identifiziert. Letztere haben wir differenziert nach semantischen und syntaktischen Hässlichkeiten. Diese

beeinträchtigen die Qualität von Lernzieldefinitionen unterschiedlich stark. Entsprechend klassifizieren wir je nach Erfüllung von Best Practices bzw. Anti-Patterns einzelne Lernzieldefinitionen nach *the Good, the Bad and the Ugly*.

Im Rahmen der genaueren Analyse der betrachteten Lernzieldefinitionen wurde erkennbar, dass mit Blick auf verständliche, in ihrer Bedeutung eindeutige, kompetenzorientierte Lernzieldefinitionen durchaus noch Verbesserungspotenzial vorhanden ist. Beispielsweise beinhalten rund 10% der Lernzieldefinitionen noch Verben wie „vermittelt“ oder „befähigt“, die auf eine dozierendenzentrierte Haltung hinweisen. Des Weiteren weist ein großer Anteil der analysierten Modulbeschreibungen in den Lernzieldefinitionen kein beobachtbares Verhalten aus. Damit sind diese nicht überprüfbar.

Gerade wegen der weit reichenden Bedeutung von Lernzieldefinitionen, sowohl für die Lehrenden als Grundlage der Gestaltung der Lehre im Sinne des Constructive Alignments, als auch für die Studierenden zum Verständnis der Prüfungsanforderungen und der inhaltlichen Ausrichtung eines Faches, ist der Anspruch an die Qualität von Lernzieldefinitionen zunehmend steigend. Entsprechend besteht hier an vielen Stellen noch Handlungsbedarf – auch für unseren eigenen Wirkungsbereich.

Zum Schluss bleibt festzustellen, dass eine automatisierte Qualitätssicherung von Lernzieldefinitionen mit den derzeit verfügbaren technischen Mitteln noch sehr schwierig ist. Entsprechend wird bis auf Weiteres eine manuelle Qualitätssicherung unabdingbar sein. Auch hier hilft ein klares Verständnis von Best Practices ebenso wie von Anti-Patterns, kritische Formulierungen schnell zu identifizieren und auf effiziente Weise mögliche Verbesserungen anzugeben.

Dank

Das Autorenteam wurde gefördert durch das BMBF Förderkennzeichen 01PL16025 (Projekt "Für die Zukunft gerüstet"), im Programm "Qualitätspakt Lehre".

Literatur

[ACQUIN 2014] Akkreditierungsagentur ACQUIN e. V.: *Leitfaden für Verfahren der Programmakkreditierung*. https://www.acquin.org/wp-content/uploads/2014/06/LeitfadenProgrammakkreditierung.pdf, 2014. – zuletzt aufgerufen am 4.11.2018

[Anderson u. a. 2001] Anderson, Lorin W. ; Krathwohl, David R. ; Airasian, Peter W. ; Cruikshank, Kathleen A. ; Mayer, Richard E. ; Pintrich, Paul R. ; Raths, James ; Wittrock, Merlin C.: *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*. 1. New York : Longman, 2001

[AQAS 2016] Agentur für Qualitätssicherung durch Akkreditierung von Studiengängen (AQAS e.V.): *Qualität Transparenz Vergleichbarkeit Informationen zur Programmakkreditierung*. 9. Auflage. https://www.aqas.de/downloads/AQAS-Broschuere.pdf, 2016. – zuletzt aufgerufen am 4.11.2018

[Arnold u. a. 1999] Arnold, R. ; Krämer-Stürzl, A. ; Siebert, H.: *Dozentenleitfaden. Planung und Unterrichtsvorbereitung in Fortbildung und Erwachsenenbildung*. Berlin : Cornelsen, 1999

[Biggs u. Tang 2011] Biggs, J. ; Tang, C.: *Teaching For Quality Learning At University*. McGraw-Hill Education, 2011 (SRHE and Open University Press Imprint). – ISBN 9780335242757

[Bischoff u. a. 2017] Bischoff, M. ; Boentert, A. ; Pernhorst, C.: *Module entwickeln und beschreiben*. https://www.fh-muenster.de/hochschule/qualitaetsentwicklung/downloads/170914_Modulbeschreibung_Druckfassung.pdf, 2017. – zuletzt aufgerufen am 4.11.2018

[Bloom u. a. 1956] Bloom, B. S. ; Engelhart, M. B. ; Furst, E. J. ; Hill, W. H. ; Krathwohl, D. R.: *Taxonomy of educational objectives: The classification of educational goals*. New York : David McKay Company, 1956

[Cursio u. Jahn 2015] Cursio, M. ; Jahn, D.: *Formulierung kompetenzorientierter Lernziele auf Modulebene*. https://www.nat.fau.de/files/2015/12/03-Leitfaden-Leitfaden-zur-Formulierung-kompetenzorientierter-Lernziele-auf-Modulebene-NatFak-und-FBZHL.pdf, 2015. – zuletzt aufgerufen am 9.11.2018

[DiZ 2018] DiZ: *Lernzieldefinition des Kurses „Rechtsgrundlagen für die Lehre an Hochschulen“*. https://diz-bayern.de/programm/termine-und-buchung/details/4-diz-termin?xref=161503:rechtsgrundlagen-fuer-die-lehre-an-hochschulen, 2018

[FH Furtwangen 2013] Hochschule Furtwangen University, Prorektor für Lehre und Studium/ Stabsstelle Qualitätsmanagement: *Modulbeschreibung und Formulierung von Lernergebnissen*. http://findo.hs-furtwangen.de/pub/QM_Board/HFU_Leitfaden_Modulbeschreibung.pdf, 2013. – zuletzt aufgerufen am 4.11.2018

[FH Münster 2018] FH Münster: *Modulbeschreibung „Höhere Programmierkonzepte“*. https://www.fh-muenster.de/eti/downloads/module/Modulhandbuch_Informatik_2012-06-20_Ueberarbeitet_2017.pdf, abgerufen am 27.11.2018, 2018

[Gröblinghoff 2013] GRÖBLINGHOFF, F.: *nexus impulse für die Praxis Nr. 2: Lernergebnisse praktisch formulieren*. 1. Auflage. https://www.hrk-nexus.de/fileadmin/redaktion/hrk-nexus/07-Downloads/07-02-Publikationen/Lernergebnisse_praktisch_formulieren_01.pdf, 2013. – zuletzt aufgerufen am 4.11.2018

[Hauer 2011] HAUER, E.: Wird dumm geprüft, wird dumm gelernt – Plädoyer für den Einsatz anwendungsorientierter Prüfungsaufgaben im Hochschulbereich. In: *Magazin erwachsenenbildung.at* (2011), Nr. 12, S. 10.1–10.10

[Hollender 2010] HOLLENDER, D.: *Formulierungshilfen für Modulhandbücher – Handreichung zur Verstärkung der Kompetenzorientierung*. https://www.intern.tu-darmstadt.de/media/dezernat_ii/ordnungen/Handreichung.pdf, 2010. – zuletzt aufgerufen am 13.11.2018

[HS Fulda 2018] HS Fulda: *Modulbeschreibung „Programmierung 1“*. https://www.hs-fulda.de/fileadmin/user_upload/Unsere_Hochschule/Hochschulrecht/Pruefungsordnungen_der_Fachbereiche/Angewandte_Informatik/AI_BSc_AI_2017-06-21.pdf, abgerufen am 27.11.2018, 2018

[HS Kempten 2018] HS Kempten: *Modulbeschreibung „Softwaretechnik 1“*. https://www.hs-kempten.de/fileadmin/fh-kempten/E_I/inf_bsc/pdf/20180727_Modulhandbuch_IF_-_Beschlussvorlage.pdf, abgerufen am 27.11.2018, 2018

[HS München 2018] HS München: *Modulbeschreibung „Softwareentwicklung 1 (IB)“*. https://w3-o.cs.hm.edu:8000/public/module/226/, abgerufen am 27.11.2018, 2018

[Kennedy 2008] KENNEDY, D.: *Lernergebnisse (Learning Outcomes) in der Praxis – Ein Leitfaden; Deutsche Version: T. Mitchell, V. Gehmlich, M. Steimann*. 2008

[Mager 1972] MAGER, R.: *Lernziele und programmierter Unterricht*. 35. Weinheim : Beltz, 1972

[Metzger u. Nüesch 2004] METZGER, C. ; NÜESCH, C.: *Fair prüfen: Ein Qualitätsleitfaden für Prüfende an Hochschulen*. 2004

[Rechenberg 2006] RECHENBERG, P.: *Technisches Schreiben: (nicht nur) für Informatiker*. Hanser, 2006. – ISBN 9783446406957

[Schneider 2002] SCHNEIDER, W.: *Foliensatz “Einführung in die Wirtschaftspädagogik”*. Wien, 2002

[Terhart 2005] TERHART, E.: *Lehr-Lern-Methoden*. 4. Weinheim : Juventa, 2005

[Toutanova u. a. 2003] TOUTANOVA, Kristina ; KLEIN, Dan ; MANNING, Christopher D. ; SINGER, Yoram: Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, 2003 (NAACL ’03), S. 173–180

[TUM 2016] Technische Universität München: *Wegweiser zur Erstellung von Modulbeschreibungen, Version 3*. https://www.lehren.tum.de/fileadmin/w00bmo/www/QM_Handbuch/Dokumente/Wegweiser_Modulbeschreibungen_Version3_Stand_Maerz_16.pdf, 2016. – zuletzt aufgerufen am 4.11.2018

[Uni Würzburg 2013] Julius-Maximiliansuniversität Würzburg: *Output-Orientierung und Kompetenzformulierung im Bologna-Prozess*. https://www.uni-wuerzburg.de/fileadmin/39030000/ZiLS/Material/Kompetenzorientierung/Kompetenzformulierung_15.10.2013.pdf, 2013. – zuletzt aufgerufen am 4.11.2018

[UZH 2008] *Dossier Unididaktik – Lernziele formulieren in Bachelor- und Masterstudiengängen*. https://www.hrk-nexus.de/fileadmin/redaktion/hrk-nexus/07-Downloads/07-03-Material/DU_Lernziele_11_08.pdf, 2008

[Weinert 2001] WEINERT, F.E. (.: *Leistungsmessung in Schulen*. Weinheim : Beltz, 2001

Teaching Rationale Management in Agile Project Courses

Anja Kleebaum[1], Jan Ole Johanssen[2], Barbara Paech[1], and Bernd Bruegge[2]

[1]Heidelberg University, Heidelberg, Germany
[2]Technical University of Munich, Munich, Germany

kleebaum@informatik.uni-heidelberg.de, jan.johanssen@tum.de,
paech@informatik.uni-heidelberg.de, bruegge@in.tum.de

Abstract

Rationale management is beneficial since it supports decision-making and prevents knowledge vaporization. To apply rationale management, developers need to know how to systematically capture rationale and how to exploit the documentation. We believe that teaching these skills to students further integrates rationale management into the daily work of developers and has positive effects on both the software development process and on the quality of the software. In this paper, we report on a lecture on teaching rationale management to students. In this lecture, students are introduced to a rationale model, to capture and exploitation methods, and tool support for rationale management. The goal is to motivate them to apply rationale management. We present the students' results as well as their attitude and feedback towards the applied methods. Further, we sketch how rationale management will be applied during the semester.

1 Introduction

Developing software means to continuously solve issues and to make decisions. Developers possess knowledge about decisions, the issues they solve, alternatives, and their justifications. This knowledge is called decision knowledge or rationale. The success of a development project strongly depends on the decision-making abilities of the developers and other relevant stakeholders. Rationale management supports explicit decision-making by capturing rationale and by using the documentation [7]. Developers are said to be reluctant to capture rationale systematically [1, 14, 18]. We believe that this can be alleviated by teaching developers how to capture rationale and by raising their awareness for the benefits of rationale management.

As part of our previous work, we developed methods and tool support to integrate rationale management into agile development processes, in particular into continuous software engineering [11, 12]. In this paper, we report on a lecture on teaching rationale management to university students. The goal of this lecture is to teach a rationale model to students, to introduce them to methods and tool support for rationale management, and to motivate them to apply rationale management in the future. We gave this lecture as part of an agile multi-project course at the Technical University of Munich, in which student teams work on different software projects over the period of one semester. The projects are initiated by industrial customers [4]. As a default set of tools, the students use the issue tracking system JIRA and the wiki system Confluence. During the lecture, we presented our methods on rationale management to the students and they applied parts of our tool support. We asked students for their feedback on the presented tools and to perform exercises.

This paper is structured as follows. In Section 2, we outline background knowledge on rationale management and the agile multi-project course. In Section 3, we present our teaching material including six exercises, and the course of the lecture. Then, we summarize the students' feedback, present the exercise results, and discuss lessons learned. In Section 4, we describe plans on how to evaluate rationale management during the agile project course. Section 5 lists related work and Section 6 concludes the paper.

2 Background

This section introduces the rationale model we use, basics about issue tracking and wiki systems, our previous work on integrating rationale management into agile software development, and a description of the agile multi-project course that the lecture is part of.

2.1 Rationale Models

A rationale model represents knowledge in the same way a system model represents a system. Rationale can be modeled as a graph of rationale elements and edges that represent the elements' relationships. There are various models for building such graphs of rationale. In general, these models differ in the types of elements and edges that they allow. For example, an advanced model is the decision documentation model,

Emoji	Name	Indicating Phrases
	Issue	I have a question ... How should ... ..., any suggestions? We need to discuss how ...
	Alternative	I { suggest | propose } ... One { option | proposal } is ... What { about | do you think } ...
	Pro	The { advantages | pros } are ... I { like | prefer } it because ... I agree with user ...
	Con	The { disadvantages | cons } are ... I don't like it because ... I disagree with user ...
	Decision	Let's do ... We decided ... The best option is ...

Table 1: Types of rationale elements, their representing emoji, and indicating phrases adapted from [5].

which makes fine-grained differences in element types such as implications resulting from a decision or constraints that need to be considered [9]. There could be various types of relationships between rationale elements, for example, a decision can *solve* an issue or it can also *lead to* a new issue.

In order to teach rationale management, we use an easy-to-learn model to represent rationale. This model covers five types of rationale elements: *issue*, *alternative*, *pro-* and *con-argument*, and *decision* (Table 1). In our model, we distinguish three types of relationships, i. e., edge types. The *relates to* relationship is the default edge type. Only for arguments different types are used: A pro-argument *supports* an alternative or the decision, whereas a con-argument *attacks* an alternative or the decision.

2.2 Issue Tracking and Wiki System

Developers can capture rationale in various documentation locations, for example, in the issue tracking [3, 10, 18] or the wiki system. Issue tracking systems are widely applied to store requirements, bug reports, as well as development tasks. They contain various information types such as functionality or quality requests and *as-is* descriptions [17]. Wiki systems are collaborative writing tools that can be applied for many purposes, for example, for meeting management and requirements elicitation [19]. For a lecture on rationale management, we assume that students use the issue tracking system JIRA and the wiki system Confluence.[1] JIRA and Confluence are commercial systems that provide free licenses to universities. Both systems can be extended with plugins.[2]

[1] https://atlassian.com/software
[2] https://developer.atlassian.com

2.3 Continuous Rationale Management

Agile processes support lightweight, flexible, and continuous software development. Rationale management integrates well into such processes, yet, it should be easy to apply, i. e., developers should need as little effort for it as possible. Capturing and exploring rationale should be non-intrusive, which means that developers should be able to perform rationale management as part of their daily practices rather than having to change their development context. For example, developers should be able to capture and explore rationale simultaneously with performing development tasks, implementing requirements, or committing code.

In order to fulfill this requirement of non-intrusiveness, we develop tool support that directly integrates into the development tools [12]. In particular, we integrate our tool support into the issue tracking system, version control system, wiki system, chat system, and the integrated development environment. We refer to our tool support as ConDec, standing for the continuous management of decision knowledge. The ConDec tool support is available online.[3]

While ConDec comprises features to trigger developers in capturing and exploring rationale [11], in this paper, we focus on the basic infrastructure necessary for capturing and visualizing rationale as a graph. Regarding the ConDec tool support, the students get to know and apply the ConDec JIRA plugin, i. e., the tool support for the issue tracking system JIRA.

The ConDec JIRA plugin supports different documentation locations of rationale, e. g., JIRA issues and comments. On the one hand, rationale elements can be captured as JIRA issues with special types for issue, alternative, argument, and decision. Note the difference between JIRA issue as an abstract container and the concrete issue as the rationale element. JIRA issue links are used to link the rationale elements with each other and also to JIRA issues of other types such as scenarios or tasks. On the other hand, rationale can be captured as part of the comments of JIRA issues.

2.4 The iPraktikum

The iPraktikum is a multi-project course in which up to 100 students work in eight to ten teams on real problems provided by an industry customer [4].

In particular in the first half of the semester, the practical character of the course is supported by theoretical, yet interactive lectures. During these lectures, the students learn the basic concepts of agile development, release and merge management, modeling, and usability engineering. Recently, we added a lecture on rationale management. During the iPraktikum, the students apply latest tools and frameworks that are used in real industry projects. Moreover, they get to use the results of latest research projects, which might

[3] https://github.com/cures-hub

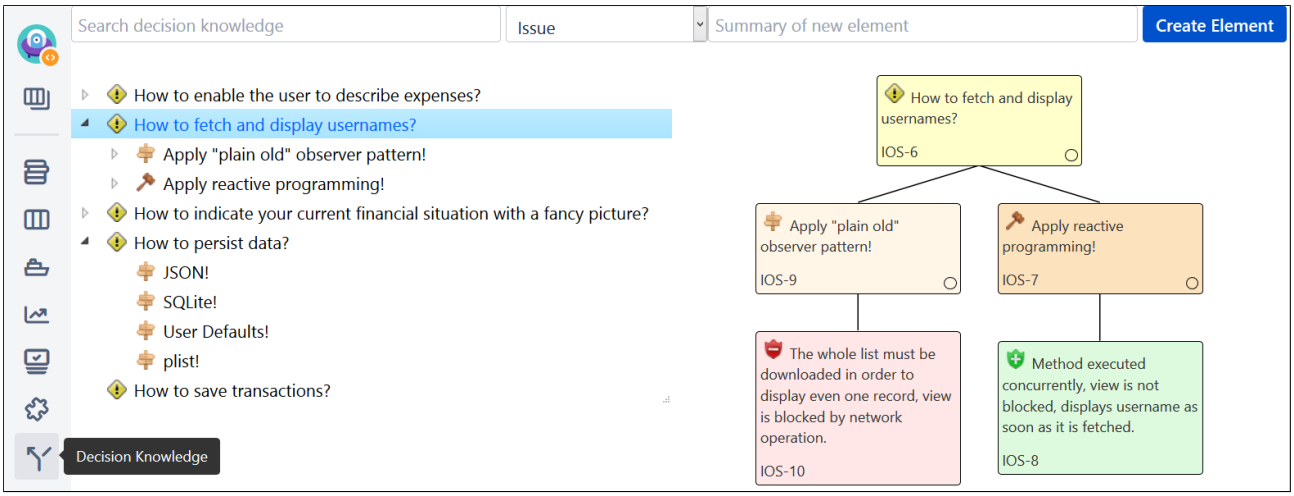


Figure 1: The decision knowledge view as a separate view for performing rationale management in JIRA.

find their way into the development process in the future. This prepares the students for their future use.

At the time of the rationale management lecture, they have already been introduced to JIRA and Confluence. Regarding JIRA, they have at least one month of experience. This knowledge is imparted through various channels: (a) a development introduction course in which the students are required to track the progress of their work via JIRA, (b) the work within their teams, and (c) a course-wide lecture on managing the backlog, on what JIRA issues are, on how to close, move, and work with them in sprints. Regarding Confluence, we provide the students with a meeting management introduction at the beginning of the course. This is handled by the coaches of a team, a special role within the team that is fulfilled by an experienced student and which is similar to a scrum master. The coach can decide on what to present, however, we provide a guideline that contains the major aspects of using Confluence. In particular, this relates to managing the team agenda, i. e., (a) how does a team meeting schedule look like?, (b) which roles are present during each meeting?, (c) what are guidelines, i. e., what to provide as the stand-up information and when to use it?

3 Lecture on Rationale Management

In this section, we describe the lecture, the results of its first instantiation, and then discuss these results and our lessons learned.

3.1 Preparation and Introduction

The lecture is designed to last 90 minutes. Students are grouped into teams and require a web-connected device with access to the internet.[4] During the lecture, three systems are needed: JIRA, Confluence,

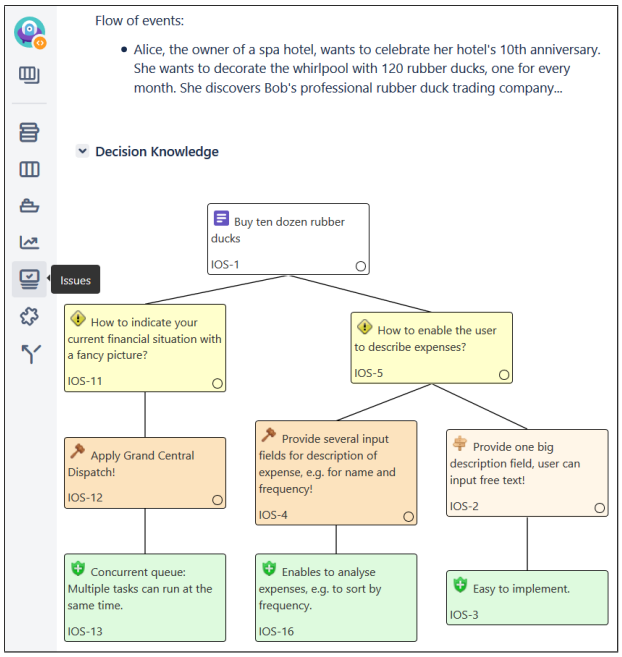


Figure 2: The JIRA issue view including the interactive rationale tree.

and the instant messaging services Slack[5]. Further, the ConDec JIRA plugin[6] needs to be installed in JIRA and be enabled for the specific projects that the student teams work in. Rationale elements that can be explored by the students need to be added to the respective JIRA projects, e. g., the elements shown in Figure 1 and Figure 2. Slack is used as a communication tool between the instructors and students. In particular, polls can be created with the Polly Slack app[7] through which students participate during the lecture.

[4] An alternative would be to run the servers locally and to give students access to the intranet (not possible for Slack).

[5] https://slack.com
[6] https://github.com/cures-hub/cures-condec-jira
[7] https://polly.ai

The first part of the lecture covers background information about rationale management, such as its definition, expected benefits, and the rationale elements. We advise students to use certain phrases when talking about and capturing rationale (Table 1). Further, we recommend to phrase issues as questions ending with a question mark and to end alternatives with an exclamation mark.

3.2 Capturing, Visualizing, and Filtering Rationale in JIRA

In the second part of the lecture, the students are introduced to the JIRA ConDec plugin including the JIRA issue types for rationale elements (Figure 3).

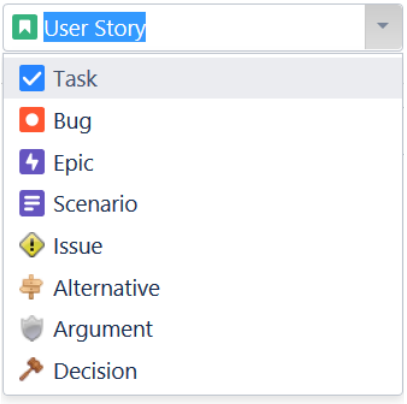


Figure 3: JIRA issue types available in the project.

Two views for rationale management are shown to the students: the decision knowledge view (Figure 1) and the JIRA issue view (Figure 2). The decision knowledge view is a separate view that holds all rationale elements and their links for the given project. In this view, a user can select single rationale elements and visualize the graph of rationale as a tree, called rationale tree. In the JIRA issue view, rationale attached to JIRA issues can be explored. For example, Figure 2 shows the rationale tree for a scenario.

Then, the instructor demonstrates how to create and link rationale elements shown on the right side of Figure 1. Afterwards, the students gather in teams and perform the first exercise:

Exercise 1 *Gather your team, open your JIRA project, and find the scenario already documented in the project. Answer the question: How many issues are linked to the scenario?*

The students can answer the question via a poll. In our example, the correct answer is that two issues are linked to the scenario (Figure 2). The next exercise is to discuss the content of the solution proposals, i. e., the alternatives and decisions.

Exercise 2 *Answer the question: Which of the proposed alternatives and decisions focus on requirements, which of them on implementation?*

In our example, the decision and alternative on the left are more requirements-related, whereas the decision on the right is implementation-specific (Figure 2). With this exercise, the students should learn that rationale can be captured for all steps in the software engineering process, including requirements elicitation, implementation, deriving test cases, and when processing user feedback.

Now, the students will make their first experience in capturing rationale:

Exercise 3 *Link the existing issue "How to save transactions?" to the scenario.*

This issue is already part of the JIRA project (Figure 1) but not linked to the scenario yet. The students can link the issue via a context menu on the scenario node (root node in Figure 2).

The students realize that graphs of rationale can become large and complex. Therefore, filtering is important. The next exercise addresses filtering:

Exercise 4 *Filter the element types so that only the scenario and decisions are shown in the rationale tree.*

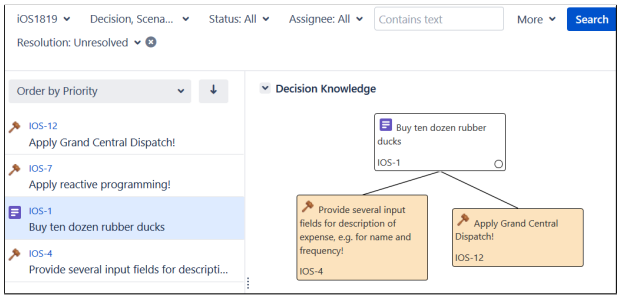


Figure 4: Filtered rationale tree.

Figure 4 shows the solution of this exercise. Now, students discuss rationale in their team:

Exercise 5 *Create a new issue "How to persist data?". Add the following alternatives: "JSON!", "SQLite!". Discuss and capture pro and cons of each persistence alternative in your team. Add more alternatives and make a decision.*

To solve this exercise, the students can add rationale elements collaboratively from different devices. This exercise takes about 10 to 15 minutes.

Rationale can be captured in many places. JIRA issues are just one possible documentation location to store rationale. In order to demonstrate that there are other possible documentation locations, the students are presented with capturing rationale in JIRA issue comments (Figure 6). In this lecture, this is for information only, but, as part of our future work, we integrate various documentation locations typical for continuous software engineering and support the identification of rationale elements with a supervised text classifier.

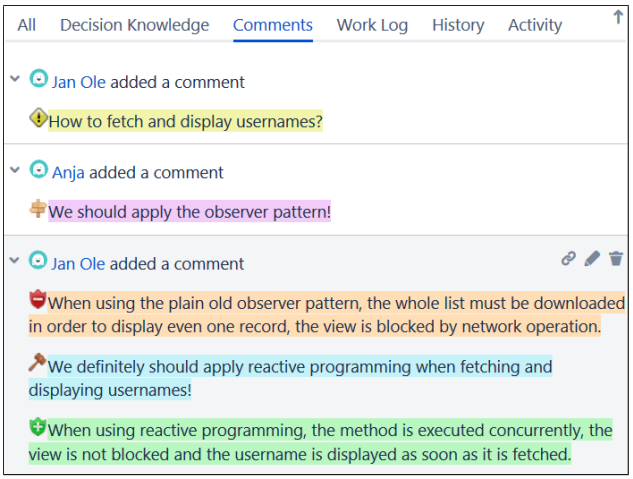


Figure 6: Explicit rationale in the comments of the scenario.

3.3 Rationale-based Meeting Management

Capturing rationale pays off during sprint meetings. The meeting agenda has an information sharing section that lists issues discussed and decisions made during the last sprint. If meeting agendas are managed in Confluence, the rationale elements captured in JIRA can be easily imported.

Exercise 6 *Gather your team, open your Confluence space and create a new page called <Rationale Lecture> (one per team). Create a sub-page called <Your Name> (every team member). Use the JIRA Issue/Filter macro to display decisions from your JIRA project. Answer the question: How many decisions do you see?*

The JIRA Issue/Filter macro is used in the exercise and the search string is expressed using the JIRA query language (JQL). The JQL string is:

```
project = <Project Key> AND
issuetype = Decision AND
created > -7d.
```

In our case, two decisions are shown.

3.4 Results

Two authors of this paper gave the lecture on November 8, 2018. In this section, we present results of this first instantiation of the lecture.

The first poll was to answer the question what team a student belongs to. With this poll, we wanted the students to get warmed up in voting and to get the number of participating students. In total, 88 students answered the poll, i. e., about 88 students participated in the lecture. This number serves as a reference point for the following, quantitative evaluation results.

Exercise 1 was answered by 64 students, i. e., 73 % of the participating students, of whom 63 gave the correct answer.

Regarding the content of the solution proposals, i. e., the alternatives and decisions, we initially asked a different question than given in Exercise 2. We asked the students to describe the difference between the alternatives and decisions. Since this question seemed to be hard to answer, we restated the question as given in Exercise 2. This exercise was verbally performed and we did not collect any results for it.

After performing Exercise 3, we asked students to assess how easy it was to link an existing issue to a scenario via a poll. They were asked to rate the statement *Linking an existing issue to a scenario is easy* with one answer from a five point Likert scale. 59 students participated in this poll, of whom 6 disagreed, 11 were neutral, and 32 agreed with the statement (Figure 5). After the lecture, we checked whether the teams correctly performed Exercise 3. In every team project, the issue was correctly linked to the scenario.

We did not collect any results for or attitudes towards Exercise 4.

For Exercise 5, we created a poll in which the students could rate the statement *Discussing rationale using ConDec is simple*. 56 students participated in this poll, of whom 7 disagreed, 13 were neutral, and 35 agreed with the statement (Figure 5).

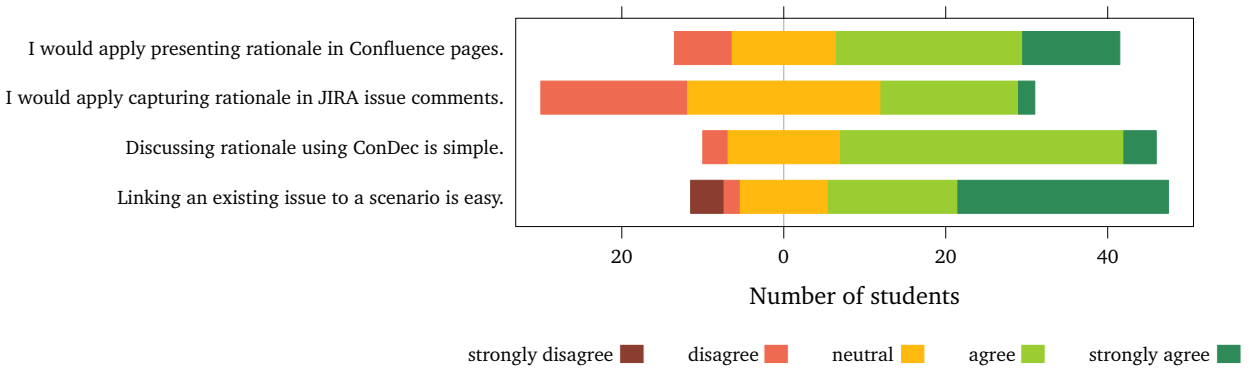


Figure 5: Students' attitude towards the presented methods and tools for rationale management.

We asked the students to provide written feedback on discussing rationale (Table 2) and we analyzed the rationale graphs. A total of 126 rationale elements were documented, i. e., a mean value of 12.6 rationale elements per team with a standard deviation of 9.4 elements. That means that each student contributed a mean value of 1.4 rationale elements. A mean value of 3.3 alternatives were documented per team (standard deviation 1.2). 61 % of the alternatives correctly ended with an exclamation mark. Seven of the ten issues correctly ended with a question mark. Four teams documented the decision. The types of the elements were correctly chosen, e. g., arguments were not accidentally classified as alternatives.

Student Feedback	Our Rationale
Deletion of elements is not possible.	Permission scheme in the JIRA project forbids deletion for non-admin users.
If there are many alternatives and arguments, it is hard to get an overview. The user needs to scroll very far to the right to see all the content.	A better graph visualization and better, easy-to-apply filter possibilities are needed.
Rationale elements should not always be a single ticket. This seems to end up in a ticket overflow. Pro- and con-arguments should be comments.	Capturing rationale elements in separate JIRA issues has the advantage that they can easily be imported into Confluence. We also develop the ConDec Confluence plugin to import rationale from comments.

Table 2: Summarized feedback provided by students.

After demonstrating that rationale could also be captured in the comments of the scenario, we asked the students to rate the statement *I would apply capturing rationale in JIRA issue comments*. 61 students participated in this poll, of whom 18 disagreed, 24 were neutral, and 19 agreed with the statement (Figure 5).

After the students performed Exercise 6, we asked them to rate the statement *I would apply presenting rationale in Confluence pages*. 55 students participated in this poll, of whom 7 disagreed, 13 were neutral, and 35 agreed with the statement (Figure 5).

A mean value of 58 students participated in the polls analyzed in Figure 5, which represents 66 % of all students.

3.5 Discussion and Lessons Learned

Clearly, the results that we collected during the lecture only represent the students' first impression about the presented methods for rationale management and the ConDec JIRA plugin. As described in the next section, we will apply a more thorough evaluation process during the semester.

The results of voting on the statements in Figure 5 lead us to conclude that the majority of the students liked applying rationale management. More important than the votes, the written feedback summarized in Table 2 encourages us to improve the visualization and filtering components of the ConDec JIRA plugin. In general, we had the impression that the students liked voting on the polls and performing the exercises, since this made the lecture more interactive [13].

Studying the rationale trees that the students created in Exercise 5, we noticed that we did not ask the students to make a decision. Thus, only four decisions were documented. As a result, we will restate the exercise for future use. The students seem to understand the elements of the rationale model (Table 1), since they correctly classified the element types in their trees of rationale.

4 Evaluation During Semester

Two teams continue to apply rationale management during the agile project course. Thus, we will evaluate the explained methods for rationale management, especially the ConDec JIRA plugin.

We introduced the role of the rationale manager. The rationale manager is responsible for checking and improving the rationale quality, i. e., they make sure that important elements are documented and that they are consistent. Further, the rationale manager imports issues and decisions important for the last sprint into the meeting agenda in Confluence. They update and add rationale elements after the meeting in JIRA. The role of the rationale manager is taken by one student per team. The role is passed on after a week to a different student, i. e., it is an interchanging role. After the students have completed the role of the rationale manager, we ask them to give us feedback by filling in a questionnaire. We derive the questions in this questionnaire by considering the variables of the technology acceptance model [16]: We consider the perceived usefulness, the perceived ease of use, and the intention to use. Next to rating statements as shown in Figure 5, it is important that the students provide detailed feedback on the features for rationale management they applied. For example, students should provide details on what they think is useful or useless, easy or difficult, and why they think so.

5 Related Work

To the best of our knowledge there is no work that reports on teaching rationale management. Thus, in this section, we present related work on applying rationale management in student projects.

While tools for rationale management have been evaluated in student projects before, e. g., in [8] and [2], the following authors especially focus on the positive effects of rationale management for the success of the student course.

Dutoit *et al*. discuss experiences with an integrated, rationale-based modeling environment in a variety of software engineering courses [6]. By applying rationale management, they aim to enhance the communication between instructor and students, to support students in reflecting their own work, and to enable the instructor to better monitor the students' progress. These are valid benefits that we also expect when students apply rationale management during the semester as part of the agile project course. Similar to their modeling environment, ConDec integrates the rationale elements with the system elements, such as requirements. In addition, we focus on linking rationale with development tasks since they represent the place where developers need to solve issues related to design and implementation.

Malloy and Burge developed the software engineering using rationale tool SEURAT_Edu as a web-based system that replaces the former SEURAT Eclipse[8] extension [15]. Like our tool support, SEURAT_Edu aims to support students in making the best decision for an issue under consideration by explicitly reasoning about design alternatives. During their evaluation, Malloy and Burge found that the students using their tool considered more alternatives and put more thought into decision-making. Similar to their tool, the ConDec JIRA plugin is web-based, which allows students to easily collaborate. Their tool integrates with learning management systems such as Moodle[9], which has the advantage that the learning management system takes care of authentication and assignment creation. In our case, this is handled by JIRA. Similar as we do in the lecture on rationale management, SEURAT_Edu enables the teacher to supply students with incomplete rationale that they are asked to complete. In addition, SEURAT_Edu enables the teacher to create a set of "solution" rationale. It displays the status to which students reached a solution, in order to encourage them during their tasks. SEURAT_Edu performs automatic error checks, e. g., whether there are issues not solved by a decision. The ConDec JIRA plugin provides a report page that lists quality metrics. We plan to integrate support to ensure the rationale quality in the development process.

6 Conclusion

We presented a lecture on rationale management that teaches students to apply a rationale model as well as methods and tool support for rationale management. The students' feedback and the exercise results let us conclude that the students comprehend the usage of the rationale model and that they are motivated to apply rationale management. To validate this first impression, a more detailed evaluation will be part of the remaining duration of the agile project course.

[8] https://eclipse.org
[9] https://moodle.de

Acknowledgements

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES project). We thank the participants of the rationale management lecture for their participation in the exercises as well as the ConDec developers, inter alia, Tim Kuchenbuch, Jochen Clormann, and Lars Tralle. Furthermore, we would like to thank Dominic Henze, Matthias Linhuber, and Florian Angermeir for their technical support and Doris Keidel–Mueller for providing valuable feedback on the paper. The emojis are created by Yusuke Kamiyamane[10] and licensed under a Creative Commons License.

References

[1] Zoya Alexeeva, Diego Perez-Palacin, and Raffaela Mirandola. Design decision documentation: A literature overview. In *Software Architecture*, volume 5292 of *Lecture Notes in Computer Science*, pages 84–101. Springer, Berlin, Heidelberg, 2016.

[2] Rana Alkadhi, Jan Ole Johanssen, Emitza Guzman, and Bernd Bruegge. REACT: An approach for capturing rationale in chat messages. In *11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'17)*, Toronto, Canada, 2017. IEEE.

[3] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. Automatic extraction of design decisions from issue management systems: A machine learning based approach. In *11th European Conference on Software Architecture (ECSA'17)*, pages 138–154, Cham, Switzerland, 2017. Springer. ISBN 978-3-319-65830-8.

[4] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. Software engineering project courses with industrial clients. *ACM Transactions on Computing Education*, 15(4):17:1–17:31, 2015.

[5] Michael Doyle and David Straus. *How to Make Meetings Work: The New Interaction Method*. Berkley, 1993. ISBN 978-0425138700.

[6] Allen H. Dutoit, Timo Wolf, Barbara Paech, Lars Borner, and Jürgen Rückert. Using rationale for software engineering education. In *18th Conference on Software Engineering Education and Training (CSEE&T)*, pages 129–136, Ottawa, Canada, 2005. IEEE.

[7] Allen H. Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech. *Rationale Management in*

[10] http://p.yusukekamiyamane.com

Software Engineering: Concepts and Techniques. Springer, 2006. ISBN 978-3-540-30998-7.

[8] Davide Falessi, Giovanni Cantone, and Martin Becker. Documenting design decision rationale to improve individual and team design decision making. In *ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, pages 134 – 143, Rio de Janeiro, Brazil, 2006. ACM.

[9] Tom-Michael Hesse and Barbara Paech. Supporting the collaborative development of requirements and architecture documentation. In *3rd International Workshop on the Twin Peaks of Requirements and Architecture*, pages 22–26, 2013.

[10] Tom-Michael Hesse, Veronika Lerche, Marcus Seiler, Konstantin Knoess, and Barbara Paech. Documented decision-making strategies and decision knowledge in open source projects: An empirical study on firefox issue reports. *Information and Software Technology*, 79:36–51, 2016.

[11] Anja Kleebaum, Jan Ole Johanssen, Barbara Paech, Rana Alkadhi, and Bernd Bruegge. Decision knowledge triggers in continuous software engineering. In *4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 23–26, Gotheburg, Sweden, 2018. ACM.

[12] Anja Kleebaum, Jan Ole Johanssen, Barbara Paech, and Bernd Bruegge. Tool support for decision and usage knowledge in continuous software engineering. In *3rd Workshop on Continuous Software Engineering*, pages 74–77, 2018.

[13] Stephan Krusche, Nadine von Frankenberg, and Sami Afifi. Experiences of a software engineering course based on interactive learning. In *15. Workshop Software Engineering im Unterricht der Hochschulen (SEUH)*, pages 32–40, Hanover, Germany, 2017.

[14] Claudia López, Víctor Codocedo, Hernán Astudillo, and Luiz Marcio Cysneiros. Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. *Science of Computer Programming*, 77(1):66–80, 2012.

[15] John Malloy and Janet Burge. SEURAT_Edu: A tool to assist and assess student decision-making in design. In *47th Technical Symposium on Computing Science Education (SIGCSE)*, pages 669–674, Memphis, Tennessee, USA, 2016. ACM.

[16] Nikola Marangunić and Andrina Granić. Technology acceptance model: a literature review from 1986 to 2013. *Universal Access in the Information Society*, 14(1):81–95, 2015.

[17] Thorsten Merten, Bastian Mager, Paul Hübner, Thomas Quirchmayr, Simone Bürsner, and Barbara Paech. Requirements communication in issue tracking systems in four open-source projects. In *6th International Workshop on Requirements Prioritization and Communication (RePriCo)*, pages 114–125, Essen, Germany, 2015.

[18] Benjamin Rogers, Yechen Qiao, James Gung, Tanmay Mathur, and Janet E. Burge. Using text mining techniques to extract rationale from existing documentation. In *6th International Conference on Design Computing and Cognition*, pages 457–474. Springer, 2014.

[19] Carlos Solis and Nour Ali. Distributed requirements elicitation using a spatial hypertext wiki. In *5th IEEE International Conference on Global Software Engineering*, pages 237–246, Princeton, NJ, USA, 2010. IEEE.

A Syllabus for Usability Engineering in Multi-Project Courses

Jan Ole Johanssen, Dominic Henze, and Bernd Bruegge
Technical University of Munich, Munich, Germany
jan.johanssen@tum.de, henzed@in.tum.de, bruegge@in.tum.de

Abstract

Usability engineering is an important activity in today's application development, which raises the need to focus on its teaching efforts. However, in contrast to theoretical concepts, learning usability engineering is most successful in a hands-on environment. Furthermore, a challenge exists in efficiently applying usability assessment techniques to carry out usability engineering over time. We developed the UE4MP syllabus, a four meeting-based teaching approach for agile multi-project courses using cross-functional teams. The syllabus enables students to gain usability engineering knowledge and experience through hands-on learning and to make use of a platform for usability engineering. The overall goal is to relate usability concepts closely to applications that are developed by the students themselves. We applied the UE4MP syllabus over the course of one semester and report on our observations and challenges.

1 Introduction

With the availability of software for a great range of platforms, reaching from desktop computers to mobile devices, usability engineering gained high importance.

Typically, software engineering classes focus only on teaching development practices and engineering theory and miss out on the usability of an application, which results in a situation in which developers design user interfaces [26]. Nevertheless, students usually have a general understanding of *usability engineering* (UE) that raises from different sources: (a) common knowledge, intention, or personal interest in the topic, or (b) lectures, that either are focused on teaching usability concepts to the full extend or discuss usability engineering as a subtopic [17]. In any case, the students usually lack a real-world scenario in which they can apply new knowledge.

Problem 1: Students are barely able to apply usability engineering in real-world applications.

In addition, many concepts require a well-defined environment to be applied, which is not easy to provide. For example, the *Thinking Aloud* protocol by Nielsen [18] might require a time-consuming setup; this does not scale and is not easy to apply in the given time frame of a semester, even when the students know how to do it. Furthermore, it requires more steps beforehand, such as creating an understanding of the feature under test, how it can be measured, and delivering the feature to the end user.

Problem 2: Usability engineering concepts cannot easily be applied by students due to high setup efforts.

To approach both problems, our hypothesis is that students require a well-aligned teaching concept and tool support to enhance the effectiveness of teaching usability engineering. Furthermore, we argue that such a teaching concept needs to be set within a project course that provides a hands-on environment to apply usability engineering concepts over a considerably longer timeframe in a practical manner as already suggested by Wohlin and Regnell [29].

We describe a teaching syllabus and present an experience report, in which we show how the teaching concept is interweaved with the usage of a usability engineering platform. We build upon the assumption that the combination of theoretical concepts and practical elements helps to make the usability engineering more tangible for students [18].

This paper makes the following contributions:

- Description of a cross-functional team for usability engineering that allows to transfer knowledge into project teams, which reduces the need do this with every individual team member.
- A teaching syllabus that can be applied in a multi-project course to promote the hands-on application of theoretical usability engineering concepts. The syllabus incorporates the use of a usability engineering platform.
- Results from the application of the syllabus over the course of a semester, which allows to derive useful insights for further teaching efforts.

The presented work and its contributions are motivated by the idea of encouraging interaction and collaboration when teaching usability engineering as suggested by [6]. Furthermore, we foster transparency by using collaborative tools and interactive teaching units that have a high dependency on applications that are developed by the students themselves.

This paper is structured as follows. Section 2 presents related work in the context of teaching usability engineering. Section 3 provides the description of a multi-project capstone course. This course forms the environment for a syllabus consisting of four meetings that are described in Section 4. Section 5 reports on an experience report which we discuss in Section 6 through highlighting observations and challenges. Section 7 concludes the paper.

2 Related Work

We follow Nielsen's advice on teaching usability engineering by "bas[ing] the course firmly in the laboratory" [18] and thus teach in applied, project-based courses to prepare students for their later careers [9, 23, 29]. Nielsen stresses that—besides learning a required set of skills through theoretical lessons—the hands-on approach is indispensable for students to not only learn and understand the concepts of usability engineering, but also to trigger a "revolutionary change in [their] attitudes" [18]. This hands-on approach is also supported by others, such as Basili *et al.* and Ghezzi *et al.*, who stress that students need to apply their theoretical knowledge in practical projects [2, 11] to benefit from them [3, 8, 10, 28].

Offering students the chance to acquire the basic skills motivated our theory sessions at the beginning of the syllabus to be followed by hands-on exercises. According to Nielsen, this experience is in particular valuable if the students investigate the usability of applications they developed on their own [18]. Chan *et al.* present research on how to integrate teaching human-computer interaction aspects into the curriculum of master classes and highlight the importance of real customers and ongoing discussions [7]. As a result, we describe a syllabus that integrates usability engineering in a project course in which students develop applications for real customers from industry.

The need for teaching usability engineering was highlighted by Perlman in 1988 [25]. The fact that he continued to work on teaching usability engineering in 1995 [26]—almost 10 years after the initial work—shows that teaching usability engineering needs both continuous refinement and adjustments over time. We achieve this by adding agile concepts and incorporating state-of-the-art technology into our syllabus.

Newer research of teaching usability engineering is presented by Ovad *et al.*, who try to get developers in an agile industry setting to perform basic usability tasks, such as A/B tests, on their own [24]. Bruun *et al.* use a similar approach in an industrial setting to teach developers usability evaluations. With only three participants, the level of generalizability is limited, but nevertheless the results of developers quickly learning the core concepts of usability engineering look promising [5]. As shown by these researches and supported by others [20, 21], experienced developers are often the audience to teach usability engineering.

3 Course Environment

The syllabus targets an agile, multi-project course environment, the iPraktikum, which we describe in this section. We further outline the role of cross-functional teams and the CUU platform.

3.1 The iPraktikum

We describe the capstone course *iPraktikum*, which provides the environment for teaching usability engineering in agile project courses. The iPraktikum is a multi-project course which takes place every semester with 70-90 student developers who work in up to 12 project teams to create an application with a mobile context for real customers from industry solving their real problems [4]. While the mobile context is realized using Apple's iOS platform, resulting in iPhone and iPad applications, most projects are not standalone solutions, but include application servers, sensors, actuators, or wearable devices.

As shown in Figure 1, each *Project Team* consists of a *Customer* from industry, a *Project Management Team* consisting of a *Project Lead* and a *Coach*, as well as the *Development Team*. Both the project lead and the coach fulfill a role similar to a scrum master. While the project lead is a teaching assistant who is experienced in leading projects, the coach is a student who already participated as a developer. Thus, they are familiar with the infrastructure and teaching methodology. The *Developer*s are students from second Bachelor semester up to their last Master semester.

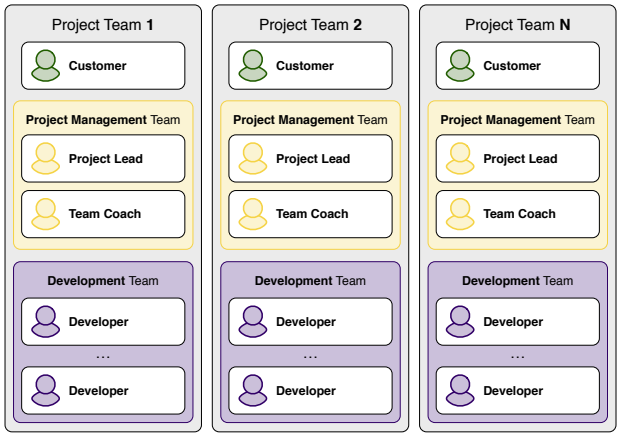


Figure 1: Overview of the iPraktikum's team structure, which consists of multiple project teams that are composed of different roles and sub-teams.

The iPraktikum fosters an event-based methodology, which enables the students to continuously interact with the customer [15]. Furthermore, the iPraktikum puts a special emphasize on the development of early prototypes and their vivid presentation to customers [31]. Over the course of multiple semesters, it has been shown that the iPraktikum's format and processes contribute to increased skills of students regarding both technical and non-technical aspects [4].

3.2 The Cross-Functional Teams

The iPraktikum uses cross-functional teams [4, 15, 31] to focus on special topics. Each of these teams is run by multiple cross-functional coaches and led by a cross-functional instructor (Figure 2). Cross-functional coaches are students experienced in the respective field. A cross-functional instructor is a teaching assistant who ensures the teaching methodology.

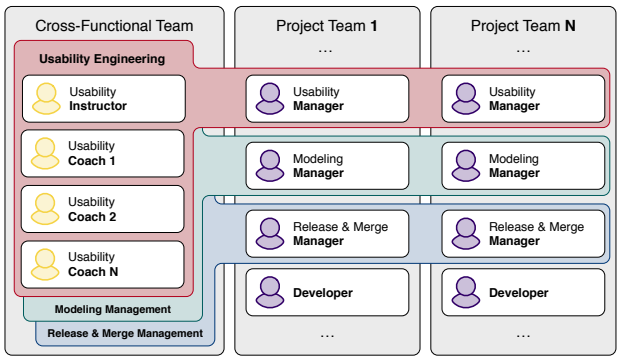


Figure 2: The iPraktikum cross-functional team structure focusing on the usability engineering team.

The cross-functional coaches work almost independently with the cross-functional teams: They elaborate on relevant topics and spread that knowledge to the cross-functional managers. These managers are developers from each team that have—in addition to their development tasks—the responsibility to share the knowledge they acquired in the cross-functional teams with their fellow team members.

While most of the work is concentrated at the beginning of the course, support and review sessions are conducted until the end of the projects to ensure an adequate quality. This approach allows the course organizers to spread knowledge from instructors to the cross-functional coaches, followed by an information sharing via the cross-functional managers to all other students of the course. Major challenges are the timing and coordination between project and cross-functional teams, as well as with course-wide events, such as course-wide lectures, or intermediate and final presentations, in which the project teams present their current status to the whole course and all customers.

So far, the iPraktikum addressed a variety of cross-functional teams. For instance, a modeling management team was established to support the project teams with creating, reviewing, and refactoring software models created during the iPraktikum [1]. Likewise, a release & merge management team helps development teams to setup the basic infrastructure for their team, including continuous integration and delivery as well as ensuring the branching model and the code review process introduced in [16]. With this work, we introduce a syllabus that extends the iPraktikum by adding a dedicated usability engineering cross-functional team.

3.3 The CUU Platform

To enable *Continuous Software Engineering* (CSE), the iPraktikum makes use of tool support—one of which is the *CUU* platform: It aims for supporting developers in understanding user behavior [12]. The overall vision of the platform is to enable usability engineering in a rapid development environment such as CSE. It is available as an open source project.[1]

CUU does not impose a particular feature definition to usability engineers: everything that is developed on a feature branch can be understood as a *feature*. We further address the definition of a feature and its meaning in the syllabus in Section 4.2. Generally, CUU analyzes whether users started, finished, or canceled the usage of a feature and visualizes this information in *widgets* [13], as shown in Figure 3.

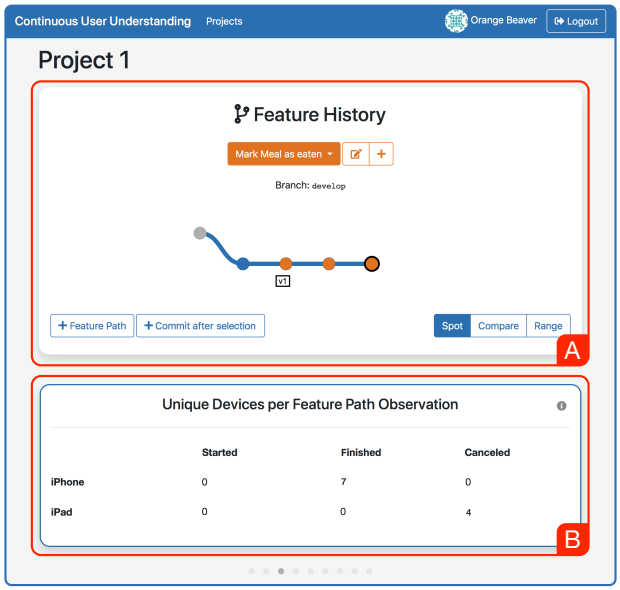


Figure 3: A screenshots of the CUU dashboard. Users select one or more commits for inspection in a graph-like representation (**A**). Related usage information is then presented in one or more widgets below (**B**).

This is enabled by the *Feature Crumbs* concept [14], a lightweight approach to mark feature specifics for detection during run-time. Feature crumbs are a single line of code, similar to breakpoints during debugging of code. Feature crumbs form the basis for referencing a variety of usability testing techniques, such as an automated implementation of the *Thinking Aloud* protocol [18]. In particular, we plan to add a widget that displays a summary of processed user feedback in relation to a feature crumb.

Besides the extract shown Figure 3, CUU hosts a *Project Overview* screen; a CUU project acts as the counterpart of a code repository. Furthermore, a *Services* screen allows UE manager to connect external services; we elaborate on this aspect in Section 6.

[1] `https://github.com/cures-hub`

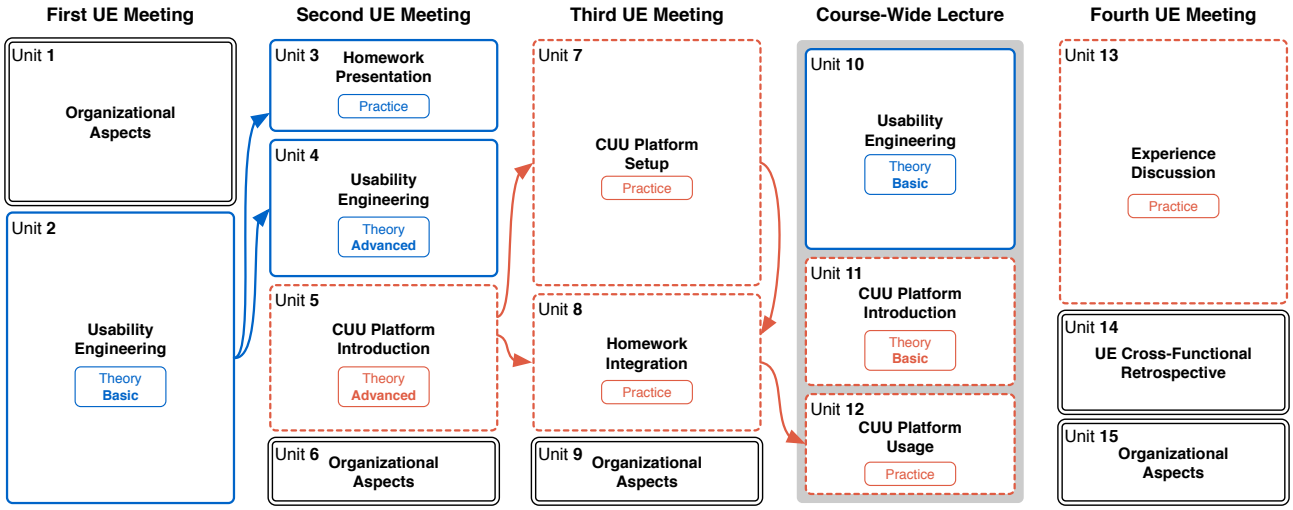


Figure 4: Visual overview of the UE4MP syllabus, a schedule for teaching usability engineering in cross-functional teams. A column represents a full meeting, which lasts for approximately 90 minutes; the UE content in the course-wide lecture might be shorter. Every box represents a Unit **U**, which encapsulates related content and is presented as one block within a meeting. The order of units suggests their actual position within the meeting and the size correlates with the time required for the unit. Arrows indicate constrains: for instance, Unit **2** must precede Unit **3** and **4**, while Unit **10** can be hold at any time. Units with blue, **single-stroked lines** address usability engineering content (4 units), while units with orange, **dashed lines** deal with the CUU platform (6 units). Units with **double-stroked lines** related to organizational aspects of the course. The **Theory** and **Practice** tags distinguish content presentation from hands-on units; theory units vary in their difficulty level: **basic** content includes overviews and topic introductions, while **advanced** content extends basic unit knowledge.

4 Teaching Usability Engineering in Multi-Project Courses

In this section, we introduce the **UE4MP** syllabus, which provides a guideline for teaching usability engineering in cross-functional teams within multi-project courses. The UE4MP reads *Usability Engineering (UE) **for** multi-project (MP) teams*, while the number 4 also represents the number of meetings during the semester. A broad overview is provided in Figure 4.

The goal of the syllabus is to step-wise introduce important concepts and to reduce the theory parts in favor of practical elements over time. A course-wide lecture, which all students of the course attend, serves as a general point of reference and milestone for the usability managers. More details about this lecture is provided in Section 5.1, while in the following, we detail every meeting with its content and goals.

4.1 First UE Meeting

The focus of the first UE meeting is the theory of usability engineering, which we base on fundamental work [18, 22]. The meeting should further clear up any organizational questions the UE managers might have at the beginning of the course.

The first UE meeting consists of many organizational aspects (Unit **1**) and starts with an introduction round of all usability managers. Furthermore, they are asked to include a brief overview of the projects they are working on. This should prepare them to provide feedback to any of the developed applications during a later point in time of the project. Hereafter, we introduce them to the role of the usability engineering manager. We summarize them as follows:

- Learn, repeat, and consolidate the general concepts of usability engineering.
- Integrate a framework for user understanding in teams' project and promote its application.
- Drive the realization of new insights that they gained from usage data and usability testing.

As the major element of the first UE meeting, the UE coaches prepare and hold a presentation of the usability engineering basics. This is roughly based on the book *Usability Engineering* by Jakob Nielsen [18]. The coaches are asked to focus the presentation on the following key aspects:

- overview of usability slogans;
- introduction to the usability engineering lifecycle;
- different forms of prototyping and their benefits and challenges, including tools and best practices;
- overview of usability heuristics to prepare the homework for the next meeting;
- how to use the Thinking Aloud protocol as one example of usability testing;
- idea of discount usability engineering.

The presentation should be interactive, and the coaches are encouraged to invite the managers to contribute ideas and descriptions while presenting the slides; usually, the managers already have a common sense of the concepts, however, they cannot formally refer to it (e.g., the Thinking Aloud Protocol.)

The homework is clarified at the end of the meeting; it reflects a first step of applying theoretical concepts to the team's applications. We ask the managers to pick and research on one usability heuristic and share their results within the second UE meeting as part of a two-minute talk. In Figure 5, we outlined a wiki page that we prepared for that purpose, in which each manager has to fill in one row.

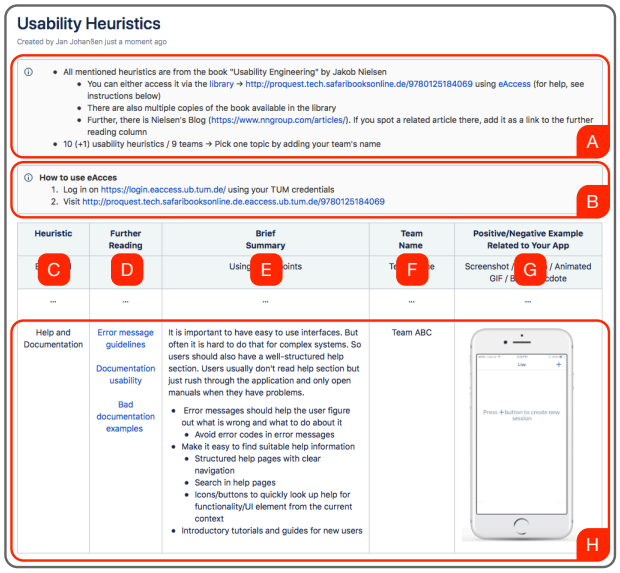


Figure 5: Wiki page created for the Usability Heuristics homework; figure modified for visualization. (**A**) Description of homework. (**B**) Help on how to access supportive material. (**C**) Heuristic name. (**D**) Link to further reading. (**E**) Brief summary of the heuristic. (**F**) Name of the team that worked on the heuristic. (**G**) Concrete example of the heuristic within the team's application. (**H**) An instance for an entry of a usability heuristic within the table.

We are particularly interested in the UE managers' results on (**G**), the application of the usability heuristic to their application. We ask them to create mockups of good and bad examples of the heuristics. Therefore, it is important to ensure that the UE managers have access to the book, either by lending a copy from the university library or having access to an online version of the book. We enriched the wiki page, on which UE managers were asked to add their homework, with a brief introduction on how to access the online library. The homework fits the setup of this course, since there are usually 10 teams and 10 usability heuristics; however, other allocations will also work.

The meeting closes with more organizational aspects, i.e., discussion on the meeting time of the next meeting and an overview of the upcoming meetings, as well as clarification of questions.

Before and after the first UE meeting, the coaches are asked to publish additional content that is relevant for the managers, such as information pages about prototyping and links to popular tools. Not only do they create and maintain these pages, they actively inform the usability managers about the availability through the chat messaging platform. The coaches further ensure that the usability managers work on their homework; this is achieved by issues that are part of the issue tracking system. This should prevent UE managers from forgetting the homework, which would reduce the learning effect for all of the others during the second UE meeting.

4.2 Second UE Meeting

The second UE meeting starts with a short presentation of the usability heuristics homework by each UE manager (Unit **3**). During the presentation of this first homework, the UE instructor facilitates the presentations of the usability managers. This idea is based on the assumption that the usability instructors have a *broader perspective* on the topic; they can assess the heuristic and know how to apply it and therefore can ask for typical problems and situations in which they occur, in case the UE manager did not mention that aspect as part of their summary. Furthermore, the UE instructors help to put the heuristic into context, e.g., by noticing relationships between them. Overall, the goal is to encourage discussions about the topics and thereby sharpen the usability managers' senses and understanding for the topic.

As part of Unit **4**, the UE coaches again provide a theory session on other usability engineering topics. This time, however, the information is notably shorter (approximately half of the time invested as in Unit **2**), focusing on the following topics:

- small repetition of the topics of Unit **2**;
- an introduction to scenarios as an instrument;
- an open discussion on the definition of a feature;
- common practices, with minor focus on UE testing approaches different from Thinking Aloud and usability evaluation through heuristics.

The second and third aspects of these topics are the most important ones. Typically, the UE managers are familiar with scenarios as a way to capture and describe requirements of a system. They also receive a repetition of this topic in a proceeding course-wide lecture, which is not solely focused on usability engineering. However, in this unit, we focus on the usage of scenarios for usability engineering [19]. This prepares their understanding of feature crumbs as described in Section 3.3. The open discussion intends to strengthen their understanding of features, how they might be represented, and to prove the point that the feature understanding is often a question of definition. They are defined by the way they are managed, e.g., user stories, scenarios, epics, and that they vary in size as well as other characteristics, such as the involved systems, stakeholders, criticality, and more.

The last major unit of the second UE cross-functional meeting is an introduction to the CUU platform (Section 3.3) following these three aspects:

- Present the platform's functionality theoretically.
- Introduce the feature crumbs concept, in particular walk through the feature path and status. Feature paths represent a concatenation of feature crumbs [14], which allows statements about the feature state using the *Unique Devices per Feature Path Observation* widget as shown in Figure 3.
- Explain the platform's goals and align them with the previously presented approaches for usability testing and evaluation. In particular, this includes an introduction of how feature crumbs can be used to perform *Heuristics Evaluation* [18], based on the knowledge that the usability managers gained through the homework.

To be able to respond to the managers' questions about the platform and the crumbs, we ask the coaches to carefully inform themselves using relevant literature [13, 14]. We also provide a short, informal summary of the two papers in form of a wiki page.

We close the meeting with Unit **6**, organizational aspects, which, for the most part, introduces the homework: For the next meeting, the UE managers are asked to prepare a feature path of a specific scenario in their application as a preparation for the third UE meeting. Therefore, we provide a wiki page, as depicted in Figure 6, which we ask every UE manager to copy into their team's space and provide a URL for reference in a shared table.

The managers start by creating a table that is similar to a formal representation of a scenario as shown in Figure 6 (**A**). Each row describes a feature crumb, split by its name, a short description and the rationale of the crumb, and under which circumstances the crumb is triggered. Then, we ask them to add comments to their source code where they think the individual crumbs from the table should be triggered. We use the comments as a first step toward the actual tracking of features. This *dry run* requires to put a focus on designing the feature, rather than starting with the tool that is able to track the execution of the feature. We provide an example of how we expect this comment to look like in Figure 6 (**B**); we put an emphasize on the naming, since it should follow the exact same spelling as in Figure 6 (**A**). This should highlight the requirement that the same name of one row needs to be exactly spelled as in the code. The same holds true for the JSON file that needs to be provided in Figure 6 (**C**). Here, the UE managers design a machine-readable version of the feature representation that they previously described as a wiki page. As with the comment-styled crumbs, this prepares an easy transition into using the tool in the next meeting.

4.3 Third UE Meeting

The third UE meeting is mainly hands-on. It enables each team to use the platform independently. We provide step-wise guidance on how to setup the platform and integrate the needed framework for an example

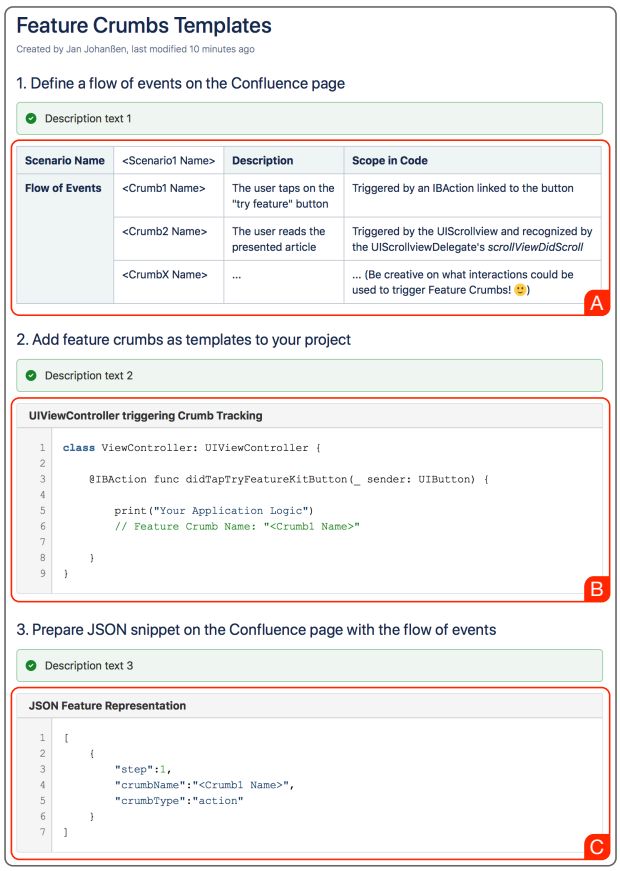


Figure 6: Wiki page created for the feature crumbs homework; figure modified for visualization, i.e., the description text has been removed. (**A**) List of crumbs to describe the UE managers' rationale when designing the crumbs. (**B**) Example of how to add the crumbs on a code level. (**C**) A JSON file that allows them to formally specify the feature path.

feature (Unit **7**), followed by the integration of their first own application-related feature (Unit **8**) that they prepared in their homework from the previous meeting. As a result, Unit **7** forms the basis for Unit **8**.

As this meeting brings results from multiple units, synchronization becomes a major challenge when preparing this meeting. The teams require a code basis which can be released and run on a device; this puts out requirements for both, the code and the workflow for its processing. Furthermore, in practical terms, the UE managers require the hardware (computer and mobile device) to run the tasks described in the following to setup and use the platform.

For Unit **7**, we prepare seven wiki pages that describe everything required to set up the platform and make use of it. Since they contain potential pitfalls, we walk the UE managers through them step by step:

Step 1 *Login and Navigate to Project.* Since the managers have never used the platform before, it needs to be ensured that they know where to find the platform and how to access it.

Step 2 *Define new Feature and link Feature Branch.* This step ensures that the UE managers adhere to the iPraktikum's development process: Every feature and its related branch is based on an issue. Therefore, we repeat the steps to formally create a branch. Hereafter, the managers need to switch back to their CUU project and create a feature representation in the platform. This can be done by providing a feature name and the initial commit on which the branch is based on.

Step 3 *Initialize Client in Code Project.* The CUU framework requires a client-side software development kit (SDK) that observes the execution of the application. The managers integrate this SDK into their projects. They need to register the SDK with a *Tracking Token* and *Project ID*; this information is extracted from the services screen (Figure 8) and ensures that the SDK can push information to the platform.

Step 4 *Add Feature Crumbs to Code Project.* In this step, we introduce the managers to the notation of triggering a feature crumb. At the bottom line, this is a method call with the feature crumb name as a string parameter. We ask them to seed a feature crumb called *My First Crumb* in the startup sequence of the application. Hereafter, the managers need to push all the latest changes to the remote server.

Step 5 *Add Commits to Branch.* The latest changes lead to a new feature increment, which needs to be registered with the platform. Therefore, they need to copy and paste the latest commit hash to their feature in the CUU platform.

Step 6 *Add new Feature Path.* Finally, to associate a commit with a feature path, they select the commit in the CUU platform and use a popup window to add a feature path in the JSON format. In this case, we ask them to only use a single-step path, with step information *1* and feature crumb name *My First Crumb*.

After they performed these six steps, the feature is ready for usability assessment. To test whether the setup was correct, they need to release the latest commit. We introduce them to the different widgets and how they can be harnessed to derive information about the feature's performance. By asking the managers to add only one crumb with the same name and at the same position, we eliminate any possible confounding factors that might distract the UE managers from their actual goal: setting up the feature for tracking. Only after they were successfully able to follow steps 1 to 6 and see a change in feature usage, we can be sure that everything works as expected.

To bring the focus back to their projects, we return to their prepared tasks from the homework in Unit **8**. This is usually much more complex, but of actual relevance for the UE managers. As a side effect, they get to know the inner workings of the platform even better. They learn how to add new crumbs to a feature path and what this means for the analysis, i.e., a new feature version, by replacing the prepared crumb comments with actual method calls.

At the end of this meeting, every team has created at least one feature that can be fully tracked using the CUU platform. This serves as the input for a hands-on exercise (Unit **12**) in the course-wide lecture.

The meeting is closed by another round of organizational aspects (Unit **9**). In particular, we explain to the managers how they can invite their team members to the project using a user management screen that is part of the CUU platform. As this is an important requirement for the course-wide lecture, we track this progress in order to ensure the effectiveness of the lecture. Eventually, we encourage the managers to continue using feature crumbs from this meeting on, collect and note down any observations, feedback, and lessons-learned, as a preparation for the fourth UE meeting.

4.4 Fourth UE Meeting

The fourth UE meeting serves as an opportunity to stimulate the discussion between the UE managers. The goals of this meeting are manifold, however, the focus lies on Unit **13**, in which every UE manager presents a feature in more depth. This includes the presentation of individual feature paths. Since this UE meeting is set at a later point of the project, we expect that the managers not only present the feature composition, but also report on how the tracking helped them in terms of usability evaluation and assessment. Ideally, the managers elaborate on knowledge gained from the feature crumbs observations and how it supported them to solve a problem. These in-depth descriptions of actual insights in real applications can be beneficial for the UE managers of other teams.

Unit **14** is a retrospective in which the managers provide feedback on both the work with the platform and the overall composition of the UE cross-functional role and the composition of the syllabus.

Eventually, Unit **15** closes the meeting with a set of information regarding the remaining semester, such as their responsibilities for future deliverables or how they can approach us if questions remain.

5 Experience Report

We applied the UE4MP syllabus from Section 4 during the summer term of 2018 in an instance of the iPraktikum, as described in Section 3. We followed the intention of a case study as defined by Wohlin *et al.* by "provid[ing] a deeper understanding of the phenomena under study in its real context" [30].

Overall, we set out to ensure the applicability of the UE4MP syllabus and in particular the acceptance of tool usage and theory concepts by the usability managers. Therefore, this section reports on the temporal instantiation (Section 5.1), the results of the practical units, i.e., the homework, (Section 5.2), and the number of feature crumbs usage (Section 5.3).

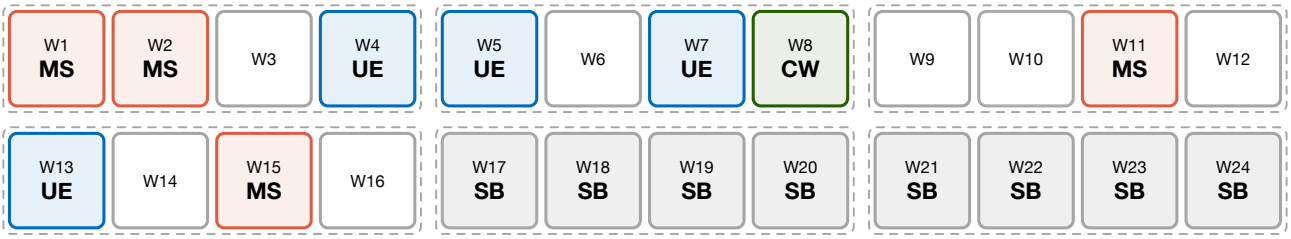


Figure 7: Chronological sequence of the iPraktikum in the summer term 2018. W1 is the first week of the semester, while W24 the last one. Supported by abbreviations, different box colors are used to map iPraktikum milestones, semester events, and UE meetings of the syllabus to semester weeks as follows. Blue boxes indicate usability engineering meetings (**UE**) and red boxes iPraktikum course milestones (**MS**). The green box highlights the usability engineering course-wide lecture (**CW**), while grey boxes relate to the semester break (**SB**).

5.1 Temporal Instantiation

To make the UE4MP syllabus transferrable to other courses, we mapped the meeting units to a typical semester of six month as shown in Figure 7.

As stated in Section 4, the syllabus builds up on cross-functional teams. These cross-functional roles, as described in Section 3, require project teams and therefore can only be assigned after the initial iPraktikum student assignment to project teams has finished. As a result, the first two weeks of the semester *W1, W2* were reserved for the iPraktikum setup, while *W3* got the projects started and the cross-functional managers of each project team assigned. In *W4*, we ran the first usability engineering meeting (Section 4.1), in which we focused on the organization, the usability engineering basics and a first homework assignment. In the following week *W5*, the second usability engineering meeting took place (Section 4.2), containing the first practical exercises and the introduction to the platform. In the third usability engineering meeting (Section 4.3) in *W7*, the tool support in form of the CUU platform was integrated in each team's project by the usability managers, followed by a first contact with the platform within each project.

After finishing the setup and explaining all concepts to the usability managers of each team, in *W8* all course participants were introduced to the concepts of usability engineering and the platform as part of a course-wide lecture. The course-wide lecture is an optional addition to the UE4MP syllabus; it intends to provide an overview of the first three usability engineering meetings. As shown in Figure 4, Unit **10** ensures a short theoretical introduction that sets the stage. In Unit **11**, we introduce the platform, how it can be used, and what it aims for. Finally, with Unit **12**, we make use of the prepared features from the third UE meeting, which allows all students to benefit from a pre-defined feature within their own application to experience the usability assessment and evaluation methods of the platform.

The timing of the course-wide lecture left the teams enough time to use the offered features until the second milestone of the iPraktikum in *W11*, which is an intermediate presentation of their work. In *W13*, two weeks after the intermediate presentation, but before the final presentation in *W15*, the fourth usability engineering meeting (Section 4.4) took place, gathering feedback for the UE cross-functional team and discussing feature paths and the corresponding feature crumbs.

5.2 Practical Unit Results

In this section, we report on the results from the homework of the usability engineering meetings, since they can serve as an indicator whether the UE managers understood the topic and were able to successfully apply the concepts.

The first homework regarding the usability heuristics (presentation in Unit **3**) was well perceived by the managers. Everyone was able to fill in the core components of the heuristic table. However, applying the heuristic to their actual application was only successfully performed by two teams; the other teams chose to use examples from well-known applications such as Microsoft Word.

In the second homework, we asked the managers to define a feature that could be tracked using feature crumbs and the platform (Section 4.2). We were able to derive more quantitative data as shown in Table 1, which allows to derive further assumptions in the discussion section (Section 6).

Table 1 shows that every team successfully defined crumbs for one of their application's features. At minimum, two feature crumbs were required, while 14 was the maximum amount of feature crumbs used to describe a feature. On average, a feature consisted of more than five crumbs. Except for one team, all teams successfully described the feature path within a JSON file. However, as it can be seen in the following section, team 4 was able to track their feature at a later point in time, which suggests that they forget to define the feature path, but did it within the platform. Out of the twelve features, ten can be considered as a proper, meaningful feature, which makes sense to be tracked and assessed for usability assessment. However, this was not the case for two features, highlighted with a red background in Table 1.

Table 1: Each team described at least one feature as part of their homework; every row depicts a feature.

Team	Number of Crumbs	Feature Path Correct?	Feature Meaningful?
1	3	Yes	Yes
2	12	Yes	Yes
3	3	Yes	No
4	14	No	No
5	3	Yes	Yes
6	3	Yes	Yes
	6	Yes	Yes
	6	Yes	Yes
7	6	Yes	Yes
8	3	Yes	Yes
	2	Yes	Yes
9	3	Yes	Yes

Team 3 implemented the feature path as three independent actions, that all started a new feature on their own. Team 4, on the other hand, described a massive feature execution, which depicted almost all of the application's features. This led to a situation in which multiple consecutive features were described as one feature, which hinders the individual assessment.

5.3 Platform Usage

To provide more quantitative data, we counted the number of features that were created by the teams throughout the semester. Table 2 shows the results.

Table 2: The number of features created in CUU separated by team; 20 features were created in total.

Team	1	2	3	4	5	6	7	8	9
Features	3	3	1	1	2	2	2	3	3

Comparing the numbers with the features that were created during the third UE meeting, which can be derived from Table 1, eight additional features were added over the course of the semester. Furthermore, we collected more data: Of the 20 features listed in Table 2, approximately 2200 feature executions were recorded successfully, more than 850 were stopped with the possibility to be successfully finished, and more than 1500 were actually canceled. In total, more than 6500 feature crumbs were triggered.

Overall, the number of features created within CUU fall short of our expectations. On average, every team created two features in CUU—one of which originated from the third UE meeting. We discuss this aspect in *The Effect of Tool Support* in Section 6.

5.4 Threats to Validity

This section briefly outlines the threats to the validity of the reported experiences centered around the four dimensions of validity [27], namely the construct, internal, external, and reliability validity.

Regarding the disparity between the intended and actual study observations, there might be the chance that the UE4MP syllabus and the introduction did not contribute to the gain of knowledge. Since there has not been a dedicated focus of usability engineering in the course before, we can assume that any information to that topic is beneficial to the students. In addition, to ensure that the concepts are straightforward, easy-to-understand, and consistent, we involved multiple instructors from other cross-functional teams to mediate the risk of ambiguous knowledge transfer.

The correlation between investigated and other factors might become visible in the way the managers made use of the taught concepts. In particular other duties, such as the actual coding or other courses, might have affected the extent to which they were working in the role of a usability manager. We tried to mediate this effect by reducing extra efforts, as well as providing help whenever needed.

The degree of generalizability of the study is low, since we can only report of one instantiation in the agile multi-project course iPraktikum. However, we argue that the presented syllabus is in general highly related to the structure of the iPraktikum, which makes generalization difficult. Still, we think that many observations and results can help other lectures to improve their teaching efforts regarding usability engineering. To be able to report more generalizable results, we plan to repeat the application of UE4MP to gain more reliable insights.

With respect to reliability, the fact that only one instructor supervised the cross-functional team might have biased the application of the UE4MP syllabus. However, since this cross-functional team was integrated in a broader course, we argue that the effect was rather low.

6 Discussion

This section discusses the results from the experience report, provides interpretations of the presented numbers, and points out challenges regarding various aspects of teaching usability in cross-functional teams.

The Effect of Tool Support. In general, we can summarize that the combination of the UE4MP syllabus and the tool support in form of the CUU platform enabled the students to perform actual usability engineering in their real-world development setting.

At the same time, the quantitative analysis of the platform usage indicates that—after an initial phase of using the platform—the teams reduced their efforts in designing feature paths and working with feature crumbs. This might have various reasons; for one, they have additional tasks which can hinder them from doing usability engineering. Other reasons might be found in the challenge of designing features, as described in the next paragraph.

The numbers suggest that we need to further support the teams in making use of the platform, pointing out the benefits which can encourage them to invest more time in usability engineering. We continue our efforts as part of our future work (see Section 7).

Usability Engineering is Difficult. A more detailed analysis of the homework illustrates that the students struggle in applying usability concepts to their own applications. Regarding the first homework, the fact that only two teams were able to create applied examples of good and bad usability heuristic cases within their own application might be an indicator that these heuristics are difficult to understand. The challenges in defining and understanding features become more obvious when analyzing the two cases highlighted as *"No"* in the column *"Feature Meaningful?"* of Table 1. The creation of a well-defined feature representation in form of feature crumbs requires effort.

Notably, in case a team provided a name for their feature, they always correctly defined the feature. This might explain the reason, why two teams created wrong feature path descriptions: They had a different mental model of a feature definition. This allows us to draw the conclusion that it is not enough to only provide a platform, which strengthens our goal of aligning the syllabus next to the platform usage.

Synchronizing Efforts. Synchronizing the UE4MP can become an issue regarding the following aspects:

- communicating the teaching materials and homework between the project leader, coaches, usability managers, and the individual team members;
- aligning the four meetings across the semester, as described in Section 5.1;
- ensuring that the teams' progress is mature enough that it fits to the UE4MP syllabus;
- connecting UE managers with other cross-functional managers, such as the release and merge managers, to ensure that they have access to the repository when setting up CUU, or are allowed to update third party repositories, which is a requirement to make the CUU platform's client-side SDK work; Figure 8 provides more description concerning this aspect.

Enable Collaboration. Besides setting up an instant messaging channel for every project team, we also created a channel for the usability engineering cross-functional team. It turns out that this is a very interesting place for discussions, which is important for usability engineering. Usability engineering is a collaborative process, which requires multiple participants, both experts and users. The students of other teams can act as proxy users; while they only have a rough idea of the topic of the other applications, they know just the right amount of details to assess the usability of the application from the perspective of a user.

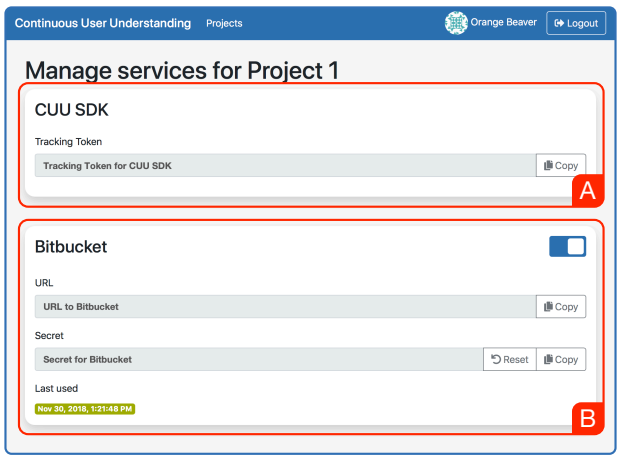


Figure 8: Screenshot of the *Services* screen of the CUU platform. Usability managers need to add the *tracking token* to their mobile application to enable the flow of usage data from the SDK's usage monitoring component to the CUU web platform (**A**). This requires a mature code basis, which might need several weeks until it is available. Furthermore, the credentials for connecting the code repository with the CUU project can be obtained from the services screen (**B**).

In general, we conclude that usability engineering can be taught better with hands-on examples, which are shared with others to collect different opinions. An instant communication channel promotes the collaboration, while wiki pages—that we used for the homework to collect usability heuristics as described in Figure 5—represent another point of interaction.

However, we noticed that many UE managers still contacted us directly in case of questions via a direct message and refrained from using the "public" channel to ask the questions to all the other students. We actively encouraged them to change this and continue to work on this in future semesters.

Acknowledging Privacy Aspects of Users. Usability engineering inherently relies on the interaction with users. With the CUU platform and SDK, the students obtain access to a tool that allows for learning more about the way a user interacts with an application. This includes, besides the ability to determine whether a feature has been started, completed, or canceled, additional knowledge sources that provide fine-grained information about the application usage. Generally, no personal data is collected in a way that would allow for conclusions toward an individual user. Within the units of UE4MP, we intend to sensitize the students for these aspects and the responsibility as a consequence thereof. In particular, we ask the students to let their end users, i.e., the customers, know about the addition of CUU before they start using it. Technical mechanisms inform the users about the data collection and provide the ability to opt out.

Providing the Environment. One challenge arises in providing the environment that is required to make use of the UE4MP syllabus. To take full advantage of the syllabus, an extensive set of tools in form of an infrastructure is required:

- an issue management system to enable tracking of tasks and development progress;
- a wiki system to share information, as well as enable a space to collaborate on the homework;
- a version control, continuous integration, and continuous deployment system that covers the full development process before the CUU platform can be utilized;
- an instant messaging service.

Providing and maintaining this environment represents a major challenge and is only feasible for large-scaled project courses.

7 Conclusion and Future Work

Usability engineering is an important activity during software engineering. However, usability engineering can only be taught successfully in a hands-on environment, using real-world projects in which students use their own applications for usability assessment. Therefore, additional, hands-on teaching approaches need to be developed to further support students besides their general software engineering curriculum.

In this paper, we introduced the UE4MP syllabus, a teaching concept based on four meetings that aims to integrate usability engineering into multi-project courses. This is enabled by establishing a cross-functional role, in which students take over the role of usability managers. Furthermore, it incorporates the CUU platform for usability engineering, which reduces the effort for setting up usability assessment activities.

We applied the UE4MP syllabus during one semester in the iPraktikum. Our observations suggest that the syllabus can be applied to teach usability engineering through cross-functional teams in a multi-project course. We were able to apply the syllabus as described and in particular the theory aspects were well-perceived by the students. Likewise, students successfully applied the CUU platform. However, it requires major efforts to teach the tool and point out benefits.

Therefore, as part of our future work, we plan to extend both the UE4MP syllabus as well as the CUU platform. Regarding the syllabus, we plan to add one-on-one meetings after or as a replacement of the fourth UE meeting, in which the usability coaches meet with the managers to discuss their work. Regarding the CUU platform, we intend to continue its development to offer more usability assessment functionalities that could be provided through additional widgets. This should increase the platform's overall usefulness and thereby increase its usage. Furthermore, we are working on simplifying the setup steps described in Section 4.3.

Acknowledgements

This work was supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution (CURES project). We thank the iPraktikum '18 students for their feedback and Simon Lang, Jan Philip Bernius, and Lara Marie Reimer for their support.

References

[1] Lukas Alperowitz, Jan Ole Johanssen, Dora Dzvonyar, and Bernd Bruegge. Modeling in agile project courses. In *MODELS (Satellite Events)*, pages 521–524, 2017.

[2] Victor R. Basili. The role of experimentation in software engineering: past, current, and future. In *International Conference on Software Engineering*, pages 442–449. IEEE, 1996.

[3] Barry Boehm, Alexander Egyed, Dan Port, Archita Shah, Julie Kwan, and Ray Madachy. A stakeholder win-win approach to software engineering education. *Annals of Software Engineering*, 6(1/4):295–321, 1998.

[4] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. Software engineering project courses with industrial clients. *ACM Transactions on Computing Education*, 15(4):17:1–17:31, 2015.

[5] Anders Bruun and Jan Stage. Barefoot usability evaluations. *Behaviour & Information Technology*, 33(11):1148–1167, 2014.

[6] John M. Carroll and Mary Beth Rosson. A case library for teaching usability engineering: Design rationale, development, and classroom experience. *Journal on Educational Resources in Computing (JERIC)*, 5(1):3, 2005.

[7] Susy S. Chan, Rosalee J. Wolfe, and Xiaowen Fang. Issues and strategies for integrating hci in masters level mis and e-commerce programs. *Int. Journal of Human-Computer Studies*, 59(4): 497 – 520, 2003. ISSN 1071-5819.

[8] David Coppit and Jennifer M. Haddox-Schatz. Large team projects in software engineering courses. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pages 137–141, New York, USA, 2005. ACM.

[9] Norman Fenton, Shari L. Pfleeger, and Robert L. Glass. Science and substance: a challenge to software engineers. *IEEE Software*, 11(4):86–95, July 1994.

[10] Christopher K. Hobbs and Herbert H. Tsang. Industry in the Classroom. In *Proceedings of the Western Canadian Conference on Computing Education*, pages 1–5, New York, USA, 2014. ACM.

[11] Paola Inverardi and Mehdi Jazayeri. *Software Engineering Education in the Modern Age: Software Education and Training Sessions at the International Conference, on Software Engineering, ICSE 2005, St. Louis, MO, USA, May 15-21, 2005, Revised Lectures*, volume 4309. Springer, 2006.

[12] Jan Ole Johanssen. Continuous user understanding for the evolution of interactive systems. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '18, pages 15:1–15:6, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5897-2.

[13] Jan Ole Johanssen, Anja Kleebaum, Bernd Bruegge, and Barbara Paech. Towards the visualization of usage and decision knowledge in continuous software engineering. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 104–108, September 2017.

[14] Jan Ole Johanssen, Anja Kleebaum, Bernd Bruegge, and Barbara Paech. Feature crumbs: Adapting usage monitoring to continuous software engineering. In *Product-Focused Software Process Improvement*, pages 263–271, Cham, 2018. Springer International Publishing. ISBN 978-3-030-03673-7.

[15] Stephan Krusche, Lukas Alperowitz, Bernd Bruegge, and Martin O Wagner. Rugby: an agile process model based on continuous delivery. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 42–50. ACM, 2014.

[16] Stephan Krusche, Mjellma Berisha, and Bernd Bruegge. Teaching code review management using branch based workflows. In *International Conference on Software Engineering - Companion Volume*, pages 384–393, 2016.

[17] Stephan Krusche, Nadine von Frankenberg, and Sami Afifi. Experiences of a software engineering course based on interactive learning. In *Software Engineering im Unterricht der Hochschulen*, SEUH '17, pages 32–40, 2017.

[18] Jakob Nielsen. *Usability Engineering*. Interactive Technologies. Elsevier Science, 1994. ISBN 9780080520292.

[19] Jakob Nielsen. Scenarios in discount usability engineering. In John M. Carroll, editor, *Scenario-based Design*, pages 59–83. John Wiley & Sons, Inc., 1995.

[20] Jakob Nielsen and Rolf Molich. Teaching user interface design based on usability engineering. *SIGCHI Bull.*, 21(1):45–48, August 1989. ISSN 0736-6906.

[21] Jakob Nielsen, Rita M. Bush, Tom Dayton, Nancy E. Mond, Michael J. Muller, and Robert W Root. Teaching experienced developers to design graphical user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 557–564, New York, NY, USA, 1992. ACM.

[22] Donald A. Norman and Stephen W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1986. ISBN 0898597811.

[23] Tom Nurkkala and Stefan Brandle. Software studio: Teaching professional software engineering. In *Technical Symposium on Computer Science Education*, pages 153–158. ACM, 2011.

[24] Tina Øvad, Nis Bornoe, Lars Bo Larsen, and Jan Stage. Teaching software developers to perform ux tasks. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, OzCHI '15, pages 397–406, New York, NY, USA, 2015. ACM.

[25] Gary Perlman. Teaching user interface development to software engineers. *Proceedings of the Human Factors Society Annual Meeting*, 32(5): 391–394, 1988.

[26] Gary Perlman. Teaching user interface development to software engineers. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pages 375–376. ACM, 1995. ISBN 0-89791-755-3.

[27] Per Runeson, Martin Host, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, 2012.

[28] Mary Shaw, Jim Herbsleb, Ipek Ozkaya, and Dave Root. Deciding what to design: Closing a gap in software engineering education. In *International Conference on Software Engineering*, ICSE '05, pages 607–608, 2005.

[29] Claes Wohlin and Björn Regnell. Achieving industrial relevance in software engineering education. In *Conference on Software Engineering Education and Training*, pages 16–25. IEEE, 1999.

[30] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN 3642290434, 9783642290435.

[31] Han Xu, Stephan Krusche, and Bernd Bruegge. Using software theater for the demonstration of innovative ubiquitous applications. In *Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 894–897, 2015.