



# GI-Edition



**Lecture Notes  
in Informatics**

**Axel Schmolitzky, Stefan Klikovits (Hrsg.)**

**SEUH 2024**

**29. Februar – 01. März 2024  
Linz, Österreich**

**Proceedings**



GESELLSCHAFT  
FÜR INFORMATIK







Axel Schmoltzky, Stefan Klikovits (Hrsg.)

## **Proceedings der SEUH 2024**

**29. Februar – 1. März 2024  
Linz, Österreich**

Gesellschaft für Informatik e.V. (GI)

## **Lecture Notes in Informatics (LNI) - Proceedings**

Series of the Gesellschaft für Informatik (GI)

Volume P-346

ISBN 978-3-88579-740-1

ISSN 1617-5468

### **Volume Editors**

Axel Schmolitzky & Stefan Klikovits

Hochschule für Angewandte Wissenschaften, Berliner Tor 7, 20099 Hamburg, Germany

Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria

axel.schmolitzky@haw-hamburg.de / stefan.klikovits@jku.at

### **Series Editorial Board**

Andreas Oberweis, KIT Karlsruhe,

(Chairman, andreas.oberweis@kit.edu)

Torsten Brinda, Universität Duisburg-Essen, Germany

Dieter Fellner, Technische Universität Darmstadt, Germany

Ulrich Frank, Universität Duisburg-Essen, Germany

Barbara Hammer, Universität Bielefeld, Germany

Falk Schreiber, Universität Konstanz, Germany

Wolfgang Karl, KIT Karlsruhe, Germany

Michael Koch, Universität der Bundeswehr München, Germany

Heiko Roßnagel, Fraunhofer IAO Stuttgart, Germany

Kurt Schneider, Universität Hannover, Germany

Andreas Thor, HFT Leipzig, Germany

Ingo Timm, Universität Trier, Germany

Karin Vosseberg, Hochschule Bremerhaven, Germany

Maria Wimmer, Universität Koblenz-Landau, Germany

### **Dissertations**

Rüdiger Reischuk, Universität Lübeck, Germany

### **Thematics**

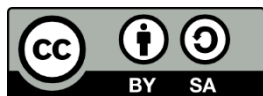
Agnes Koschmider, Universität Kiel, Germany

### **Seminars**

Judith Michael, RWTH Aachen, Germany

© Gesellschaft für Informatik, Bonn 2024

**printed by** Köllen Druck+Verlag GmbH, Bonn



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

# Vorwort

Ein herzliches Willkommen zum Tagungsband der Tagung „Software Engineering im Unterricht der Hochschulen“ (SEUH) 2024.

Die SEUH ist seit vielen Jahren das Forum im deutschsprachigen Raum, auf dem Lehrende aus Universitäten, Hochschulen für angewandte Wissenschaften sowie dualen Hochschulen ihre Erfolge, Misserfolge und Erfahrungen in der Software Engineering Ausbildung (nicht nur an den Hochschulen) vorstellen, diskutieren und gemeinsam die Qualität der Lehre verbessern. Neben inhaltlichen Impulsen rund um Lehren, Lernen und Prüfen bietet die SEUH traditionell viel Raum für Diskussion und den gegenseitigen Austausch. Viele Lehrende haben von der SEUH Impulse für ihre Arbeit erhalten.

Die SEUH 2024 fand statt an der Johannes Kepler Universität Linz in Österreich, erneut gemeinsam mit der Tagung „Software Engineering“ (SE) der Gesellschaft für Informatik. Die SEUH lebt vom regen kollegialen Austausch, der durch die gemeinsame Austragung mit der SE erneut mehr Gelegenheit erhielt. Um diesen Austausch weiter zu stärken, wurde neben den üblichen Vortragsformaten auch ein Diskussionsforum integriert, in dessen Rahmen aktuelle Themen gemeinschaftlich bearbeitet wurden.

Im Fokus der SEUH 2024 stand das Spannungsfeld zwischen etablierter Programmierlehre mit heterogenen Kohorten und neuen Wegen zum Lehren, Lernen und Prüfen einerseits und KIs als Unterstützung bei der Ausbildung im SE, aber auch bei der Softwareentwicklung selbst andererseits. Die angenommenen Papers widmen sich diesen Themen von verschiedenen Standpunkten aus. Die Einreichungen wurden dabei in vier Paper-Sessions gegliedert, in denen sie auch auf der SEUH vorgestellt und diskutiert wurden.

Wir danken allen, die zum Gelingen der SEUH 2024 beigetragen haben. Insbesondere danken wir den Autor:innen, ohne deren Beiträge die Tagung gar nicht möglich gewesen wäre, den Gutachter:innen für die reibungslose Erstellung ihrer Reviews, Guido Gryczan von der WPS GmbH für seine Vorstellung der iSAQB-Aktivitäten an Hochschulen, unserem Keynote-Speaker Hermann Sikora für seinen anregenden, fundierten und gleichzeitig unterhaltsamen Vortrag, den Sponsoren für die finanzielle Absicherung, der Johannes Kepler Universität Linz für einen wunderbaren Austragungsort und der GI e.V. mit dem Fachbereich Softwaretechnik für ihre Unterstützung.

Wir hatten zwei abwechslungsreiche Tage mit exzellenten Beiträgen aus verschiedenen Dimensionen der Softwaretechniklehre und mit vielen spannenden Anregungen und neuen Ideen.

Linz, im März 2024

Axel Schmolitzky, Stefan Klikovits

## **Tagungsleitung**

Axel Schmolitzky  
Stefan Klikovits

Hochschule für Angewandte Wissenschaften Hamburg  
Johannes Kepler Universität Linz

## **Programmkomitee**

Steffen Becker  
Ruth Breu  
Martin Fränzle  
Christian Gerth  
Claudia Hildebrandt  
Stephan Krusche  
Markus Müller-Olm  
Rick Rabiser  
Juliane Siegeris  
Veronika Thurner  
Karin Vosseberg

Universität Stuttgart  
Universität Innsbruck  
Universität Oldenburg  
Hochschule Osnabrück  
Pädagogische Hochschule Heidelberg  
Technische Universität München  
Universität Münster  
Johannes Kepler Universität Linz  
Hochschule für Technik und Wirtschaft Berlin  
Hochschule München  
Hochschule Bremerhaven...

## **Ständiges Organisationsteam**

Steffen Becker  
Axel Schmolitzky  
Veronika Thurner

Universität Stuttgart  
Hochschule für Angewandte Wissenschaften Hamburg  
Hochschule München

# Inhaltsverzeichnis

## SEUH 2024

### Session 1

**Veronika Thurner, Axel Böttcher**

<i>Herausforderungen und Angebote für heterogene Kohorten in der Softwareentwicklung . . . . .</i>	13
--	----

**Robert Ringel, Hermann Körndle**

<i>Ein kombiniertes Rahmenmodell zum Programmierenlernen . . . . .</i>	25
--	----

**Paula Rachow, Christian Rahe, André van Hoorn**

<i>Introducing Tablet-based On-Site E-Exams in a Large Software Development Course: An Experience Report . . . . .</i>	39
--	----

### Session 2

**Niklas Meißner, Sandro Speth, Julian Kieslinger, Steffen Becker**

<i>EvalQuiz – LLM-based Automated Generation of Self-Assessment Quizzes in Software Engineering Education . . . . .</i>	53
---	----

**Steffen Dick, Teresa Dreyer, Christoph Bockisch**

<i>TILE and MASS, a retrospective . . . . .</i>	65
---	----

**Bernhard M. Luedtke, Eugene Yip, Gerald Lüttgen**

<i>SWTbahn: An Embedded Software Demonstrator in Symbiosis with Embedded Software Education . . . . .</i>	77
---	----

Session 3

**Thomas R. Gross**

*Was heisst “Programmieren” im Zeitalter von LLM-basierten Programmier-Assistenten?* . . . . . 85

Session 4

**Stefan Bente, Natasha Randall, Dennis Wäckerle**

*A Conceptual Framework to Transform Coding Education in Times of Generative AI* . . . . . 93

**Victoria Geisel, Christian Schindler, Nils Stein, Stefan Bente**

*Lernräume unter Verwendung von generativen Sprachmodellen* . . . . . 105

**Autor:innenverzeichnis**

SEUH 2024





## Session 1



# Herausforderungen und Angebote für heterogene Kohorten in der Softwareentwicklung

Veronika Thurner, Axel Böttcher<sup>1</sup>

**Abstract:** In unseren Erstsemester-Kohorten werden die Studierenden immer heterogener, in verschiedenen Dimensionen. Wenn wir möglichst vielen der Studienanfänger:innen zu einem erfolgreichen Bildungsabschluss in der Informatik verhelfen wollen müssen wir unsere Lehr-Lernangebote so weiterentwickeln, dass sie den unterschiedlichen Bedürfnissen einer heterogenen Kohorte Rechnung tragen. Wir stellen eine Reihe von Ansätzen und Maßnahmen vor, die sich dafür in der Grundlagenausbildung zu Softwareentwicklung bewährt haben.

**Keywords:** Programmierausbildung, Heterogenität, Individuelle Lernpfade

## 1 Ausgangsbasis, Motivation und Ziele

In unseren Erstsemester-Kohorten werden die Studierenden immer heterogener, in verschiedenen Dimensionen. Während in den 1960-er Jahren noch weniger als 10% der Schüler:innen eines Jahrgangs eine Studienberechtigung erworben haben<sup>2</sup>, lag der Anteil Mitte der 1990-er Jahre bereits bei rund 35% und pendelt sich seit 2010 auf etwa 50% ein<sup>3</sup>. Die Übergangsquote der Studienberechtigten ins Studium liegt dabei seit Mitte der 1990-er Jahre stabil zwischen 70% und 80%<sup>4</sup>. Heute nimmt also ein zunehmend breiterer Querschnitt der Bevölkerung ein Hochschulstudium auf.

Gleichzeitig wird migrationsbedingt auch die Bevölkerung selbst zunehmend kulturell vielfältiger. Des Weiteren wachsen die Möglichkeiten der weltweiten Bildungsmobilität. All diese Faktoren (und noch viele mehr) tragen dazu bei, dass wir in unserer Lehre auf Kohorten von Studierenden treffen, die auf sehr unterschiedliche persönliche Bildungswege zurückblicken und durch vielfältige soziale und kulturelle Herkunft geprägt sind.

---

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

Gefördert durch die Stiftung Innovation in der Hochschullehre (StiL), <https://stiftung-hochschullehre.de/FoerdervertragFRFMM-455/2022>.

<sup>2</sup> [https://www.destatis.de/DE/Presse/Pressemitteilungen/2023/06/PD23\\_N036\\_12.html](https://www.destatis.de/DE/Presse/Pressemitteilungen/2023/06/PD23_N036_12.html), Pressemitteilung Nr. N036 vom 15. Juni 2023, Abbildung Studienberechtigtenquote

<sup>3</sup> <https://www.bildungsbericht.de/de/bildungsberichte-seit-2006/bildungsbericht-2022/pdf-dateien-2022/bildungsbericht-2022-kapitel-f.pdf>, Bildung in Deutschland, Hrsg.: Autor:innengruppe Bildungsberichterstattung, wbv Media (Link), Bielefeld 2022, Abbildung F2-1 Studienberechtigte

<sup>4</sup> ebenda, Abbildung F2-2 Übergangsquoten

Verschiedene Studien haben sich mit Heterogenität und Diversität in der Informatik-Ausbildung befasst (siehe dazu z. B. mehrere Beiträge im Tagungsband [De23]). Die Bandbreite der Untersuchungen reicht von der Betrachtung der verschiedenen Dimensionen von Heterogenität und der Analyse von Ursachen von Scheitern [SK22] bis zur Entwicklung von Beratungsangeboten und anderen Maßnahmen [TBH21]. Weitere Untersuchungen legen nahe, dass nicht nur die Studierenden heterogen sind, sondern auch die Erwartungshaltung der Lehrenden an die Vorfähigkeiten der Studierenden substanziell vielfältig ist [Be23].

In Summe machen die Studien deutlich, wie vielschichtig die Frage nach Heterogenität und Diversität der Studierenden ist und wie schwierig es ist, die Auswirkungen einzelner Dimensionen systematisch zu erfassen oder gar voneinander abzugrenzen. Insbesondere mit Blick auf den tatsächlichen Einfluss von Persönlichkeitseigenschaften bzw. von Schlüsselkompetenzen für den Studienerfolg wird die wissenschaftlich fundierte Datenlage sehr schnell sehr dünn – während das professorale Bauchgefühl sich im Lehr-Lernalltag hier durchaus Thesen aufstellt und diese für sich plausibilisiert. Ein Beispiel ist die These, dass Erstsemester-Studierende mit bereits abgeschlossener Berufsausbildung oft reifer, organisierter und zielstrebigere aber gleichzeitig akademisch eher aus der Übung bzw. weniger vorgebildet (insbesondere in Mathe) sind als Studierende, die direkt nach dem Abitur ins Studium einsteigen. Datenbasiert belegen lässt sich das derzeit aber nicht.

Die Beobachtungen aus der eigenen Lehre und dem Diskurs im Kollegium legen nahe, dass ein wirkungsvoller „one size fits all“-Lehransatz schwer zu finden ist. Typische Phänomene in der Erstsemester-Programmierausbildung sind, dass ein (hoffentlich großer) Teil der Kohorte an den professoralen Lippen hängt und inhaltlich anschlussfähig ist, während ein anderer Teil sich gelangweilt fragt, wann denn endlich die spannenden Themen kommen, und wiederum eine andere Studierendengruppe bereits hoffnungslos überfordert ist.

Als Abschlussurkunden-verleihende Institution haben wir die Verpflichtung der Qualitätssicherung – einen Studienabschluss gibt es also ausschließlich bei vorhandenen Kompetenzen, nicht für abgesessene Zeit. Entsprechend haben wir als Lehrpersonen für unser eigenes Wirken den Anspruch, das möglichst viele unserer Studierenden möglichst viele Kompetenzen entwickeln, fachlich wie überfachlich. Insbesondere soll also jede:r etwas für sich mitnehmen können aus unserer Lehre.

Im Folgenden analysieren wir zunächst Beobachtungen und Indikatoren, um die vielschichtigen Dimensionen der Heterogenität besser zu verstehen. Darauf aufbauend stellen wir eine Reihe von Ansätzen und Maßnahmen vor, wie wir in heterogenen Kohorten einen breiten Kompetenzerwerb der Studierenden fördern und begleiten können.

## **2 Beobachtungen und Indikatoren**

Damit wir überhaupt in der Lage sind, bedarfsgerechte Maßnahmen für eine heterogene Kohorte zu entwickeln und aufzusetzen ist es wichtig, zunächst ein Bild zu gewinnen

über die verschiedenen Dimensionen der Unterschiedlichkeiten, die uns in der Kohorte begegnen. Die nachfolgenden Zahlen beziehen sich, sofern nicht explizit anders angegeben, auf die Erstsemester-Kohorte IF1B des Bachelorstudiengangs Informatik im Wintersemester 2023/24 an der Fakultät für Informatik und Mathematik der Hochschule München. Die Vorlesungszeit begann am 01.10.2023 mit den Studieneinführungstagen, der reguläre Vorlesungsbetrieb nach Stundenplan dann am 09.10.2023.

## 2.1 Fachliche Vorfähigkeiten

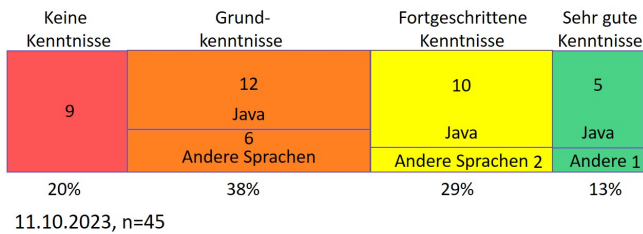


Abb. 1: Studentische Selbsteinschätzung der fachlichen Vorfähigkeiten

Um die fachlichen Vorerfahrungen zu Softwareentwicklung zu erheben haben wir in Präsenz in der ersten regulären Vorlesungseinheit nach den Studieneinführungstagen die studentische Selbsteinschätzung des initialen Kenntnisstandes erhoben (siehe Abbildung 1). Von den anwesenden 52 Studierenden haben 45 ein Votum abgegeben. Es zeigt sich eine heterogene Verteilung: Während ein Fünftel der Studierenden keinerlei Vorkenntnisse im Programmieren mitbringt attestieren sich 42% selbst mindestens fortgeschrittene Vorfähigkeiten, davon 13% sogar auf Profi-Niveau. (Das Ergebnis deckt sich weitestgehend mit dem der Vorgängerkohorte, für die im Wintersemester 2022/23 die gleiche Umfrage durchgeführt wurde.)

Die Umfrage wurde zwei Wochen später in identischer Form wiederholt. Zu diesem Zeitpunkt waren statt der initialen 52 nur noch 36 Studierende anwesend, was einem Schwund von rund 30% entspricht. Von den Anwesenden haben sich 30 an der Umfrage beteiligt.

Eine:r der Teilnehmenden hat dabei angegeben, die Selbsteinschätzung von Profi-Niveau auf Grundkenntnisse angepasst zu haben, während die Selbsteinschätzung der anderen Teilnehmenden im Wesentlichen stabil geblieben ist<sup>5</sup>. Basierend auf diesen Ergebnissen lässt sich grob zurückschließen, in welchen Niveau-Gruppen der Vorfähigkeiten bereits in den ersten Wochen des Studiums hinsichtlich Besuch der Präsenzveranstaltungen ein Schwund auftritt.

<sup>5</sup> Wir nutzten für beide Umfragen den Dienst polleverywhere.com. Dabei können die Studierenden anonym teilnehmen. Die meisten unserer Studierenden verwenden dabei aber über mehrere Sessions hinweg identische Pseudonyme.

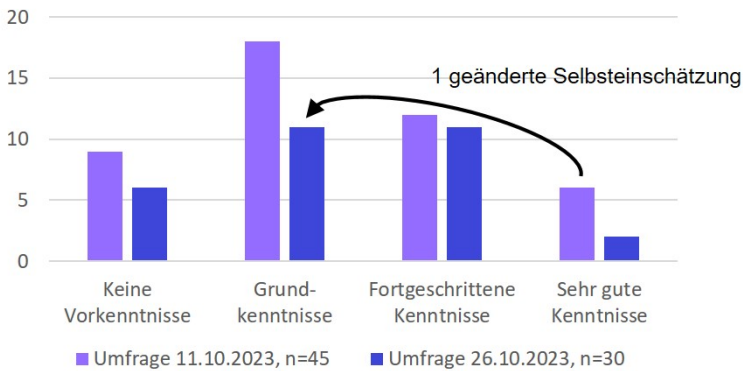


Abb. 2: Schwund differenziert nach fachlichen Vorfähigkeiten

Abbildung 2 visualisiert, dass die Teilnahme bei den Studierenden mit fortgeschrittenen Vorfähigkeiten im Wesentlichen stabil geblieben ist. Von den (wenigen) Teilnehmenden auf Profi-Niveau hat sich noch rund die Hälfte an der zweiten Abstimmung beteiligt. Von den Studierenden, die ohne jegliche Vorkenntnisse oder lediglich mit Grundkenntnissen ins Studium gestartet sind, tritt jeweils rund ein Drittel bei der zweiten Abstimmung nicht mehr in Erscheinung. Das bedeutet, dass insbesondere diejenigen Studierenden mit wenig oder gar keinen fachlichen Vorfähigkeiten frühzeitig aus der Veranstaltung aussteigen.

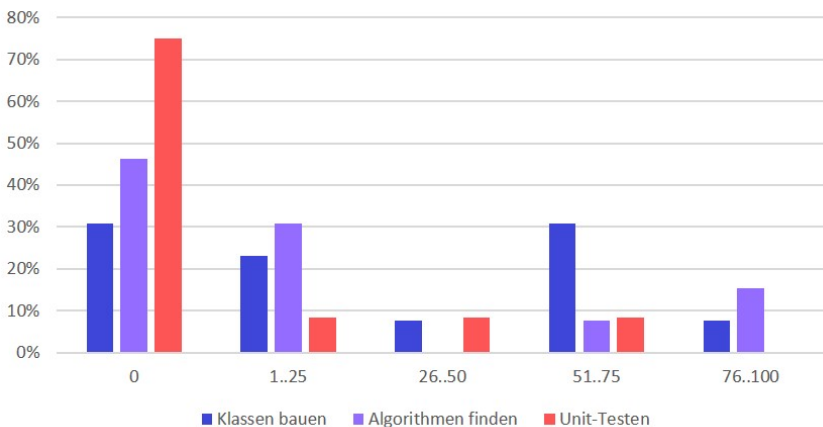


Abb. 3: Fachlichen Vorfähigkeiten differenziert nach Themenbereichen, in % der Lernzielerfüllung

Die Vorfähigkeiten selbst sind dabei auch wieder sehr heterogen, je nach Themenbereich. Abbildung 3 zeigt die Ergebnisse einer Erhebung mit der Vorjahreskohorte, in der die

Studierenden am Ende des Semesters rückblickend ihre Startfähigkeiten relativ zu den für die Veranstaltung definierten Lernzielen [Th15a] einordnen, auf einer Skala von „0 – gar nichts“ bis „100 – alles“. Dabei zeigen sich deutliche Unterschiede zwischen den Themenbereichen. Während über zwei Drittel der Teilnehmenden schon irgendwann einmal mit dem Thema „Klassen bauen“ konfrontiert war und immerhin knapp die Hälfte der Teilnehmenden schon *vor* Beginn der Veranstaltung die Hälfte der Lernziele zu diesem Thema zu erfüllen glaubt, geben dreiviertel der Kohorte an, absolut nichts über das Thema Unit-Testen zu wissen.

Auffällig ist, dass ausgerechnet beim Thema Unit-Testen im Hörsaal-Geschehen die Interaktion der Kohorte mit der Lehrperson aus Wahrnehmung der Lehrenden substanziell eingebrochen ist dergestalt, dass sich de facto niemand gefunden hat, der Fragen bzw. Aufgaben der Lehrperson an die Kohorte freiwillig beantwortet hat. Kombiniert mit der Beobachtung, dass es ansonsten im Wesentlichen immer die gleichen Studierenden sind, die in der Hörsaal-Einheit aktiv mit der Lehrperson interagieren, weckt das die These, dass sich ohnehin lediglich diejenigen Studierenden aktiv einbringen, die die gerade behandelten Inhalte schon vorher beherrscht haben – was für den Lernprozess der anderen natürlich eher kontraproduktiv ist.

## 2.2 Organisatorischer Hintergrund – dual oder nicht

Im aktuellen Wintersemester studieren rund zwei Fünftel der betrachteten Kohorte dual (konkret 21 der initial eingeschriebenen 50). Diese Studierenden haben also parallel zum Studium einen Arbeitsvertrag mit einer Firma, die das Studium finanziert mit der Intention, die Studierenden nach ihrem erfolgreichen Abschluss als Beschäftigte zu übernehmen.

Die Erfahrungen der letzten Jahre zeigen durchgängig, dass dual Studierende in der Regel ihr Studium erfolgreich zu Ende bringen. Da ziemlich genau die Hälfte der dual Studierenden bereits zu Beginn des Studiums über mindestens fortgeschrittene Programmierkenntnisse verfügt, die andere Hälfte jedoch über wenig bis gar keine Vorkenntnisse, können diese fachlichen Vorfähigkeiten nicht alleine ausschlaggebend sein für den Studienerfolg.

Aus Gesprächen mit den dual Studierenden wird deutlich, dass die begleitende Betreuung durch die Firma für die Studierenden eine wichtige Rolle spielt, und das nicht zwingend nur fachlich, sondern insbesondere auch organisatorisch. Viele der Firmen erlegen ihren dual Studierenden eine Anwesenheitspflicht in den Lehrveranstaltungen auf. Des Weiteren riskieren dual Studierende ihren Arbeitsvertrag, wenn sie Prüfungen nicht wie im Studienplan vorgegeben antreten, sondern nach hinten schieben. Entsprechend ist bei den dual Studierenden oft in erheblichem Maße Zug dahinter.

Gleichzeitig haben die dual Studierenden durch ihren Vertrag mit der Firma ein zwar moderates, aber gesichertes Einkommen über das gesamte Studium hinweg. Sie können sich also in der Vorlesungs- und Prüfungsphase voll auf das Studium konzentrieren.

### 2.3 Persönlichkeit und Schlüsselkompetenzen

Des Weiteren beobachten wir bei unseren Studierenden im Lernprozess unterschiedliche Herangehensweisen, die wir auf eine Mischung aus Persönlichkeitseigenschaften und Sozialisation zurückführen. Relevant für jeden Lernprozess erscheinen uns die beiden komplementären Eigenschaften „fleißig“ und „verstandesmäßig herangehend“, die Abbildung 4 im Sinne eines Wertequadrates nach Helwig [He48] einander gegenüberstellt. Um Programmieren zu lernen braucht es beides: den Fleiß, um z. B. das Fachvokabular sowie grundlegende Prinzipien zu lernen und den Verstand, weil jede Programmierlösung neu und einzigartig ist, damit nicht rein schematisch erstellt werden kann sondern vielmehr kreatives Problemlösen erfordert. Beide Fähigkeiten müssen also kontextsensitiv passend nach Bedarf abrufbar sein.

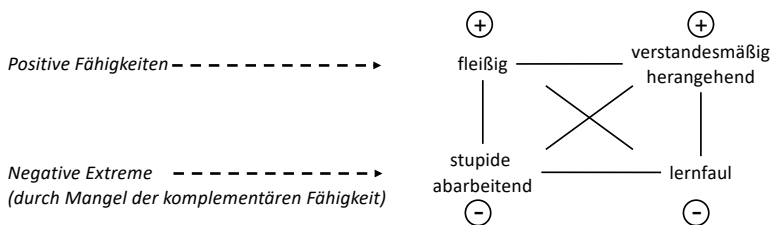


Abb. 4: Wertequadrat zu den Kompetenzen „fleißig“ und „verstandesmäßig herangehend“.

Ist eine der beiden positiven Fähigkeiten in diesem Bipol nicht angemessen ausgeprägt, so kann die andere Kompetenz ohne diese Gegenbalance ausarten zu einem negativen Extrem. Wer fleißig ist, aber den Verstand nicht nutzt, verfällt ins stupide Abarbeiten, während wer ausschließlich verstandesmäßig an Dinge herangeht, aber keinerlei Fleiß zeigt, lernfaul ist auch mit Blick auf grundlegendes Faktenwissen.

Ein hohes Maß an Fleiß bei gleichzeitigem Mangel am Einsatz des eigenen Verstandes manifestiert sich oft in guter bis sehr guter Beherrschung der Lernziel-Kompetenzen auf den Ebenen 1: Wissen und 3: Anwenden, bei gleichzeitigen Schwierigkeiten auf den Ebenen 2: Verstehen, 4: Analysieren und 6: Kreieren gemäß der Lernziel-Taxonomie nach Anderson und Krathwohl [An01]. Umgekehrt sind Indikatoren für viel Verstand in Kombination mit großer Faulheit gute bis sehr gute Kompetenzen auf den Ebenen 2: Verstehen, 4: Analysieren und 6: Kreieren, bei gleichzeitigen Ausfällen auf den Ebenen 1: Wissen und 3: Anwenden. Ebene 5: Evaluieren wiederum braucht sowohl den Verstand insbesondere für die Analyse und das kritische Hinterfragen, dabei aber gleichzeitig solides Wissen über die anzulegenden Qualitätsstandards bzw. Evaluationskriterien – hier tun sich in der Regel beide nicht ausbalancierten Varianten schwer.

In der Lehre beobachten wir immer wieder Studierende, die intellektuelles Potenzial errahnen lassen, für die der Weg ins eigenständige, kritische Denken aber eine echte Herausforderung darstellt. Teilweise führen wir das auf den Sozialisationskontext zurück, der beispielsweise von Aspekten wie Geschlechter-Stereotypen, dem in der Herkunftsfamilie bzw. Gesellschaft



geltenden Grad an Meinungsfreiheit oder dem Verhältnis von Drill zu Selberdenken in der genossenen Schulbildung beeinflusst ist – alles Phänomene, die sowohl national als auch international zu finden sind. Uns als Lehrenden muss in derartigen Konstellationen bewusst sein, wie groß und hürdenreich der Schritt zum „Selber denken macht schlau“ für diese Studierenden sein kann.

## 2.4 Herausforderungen

Aus den obigen Aspekten der unterschiedlichen fachlichen Vorfähigkeiten, organisatorischen Hintergründe, Persönlichkeitseigenschaften und Schlüsselkompetenzen ergibt sich für jede:n einzelne:n Studierende:n eine individuelle Gemengelage, die wesentlichen Einfluss darauf hat, wie gut jede:r Einzelne mit dem eigenen Lernprozess vorankommt. Auch das haben wir für die Erstsemester-Kohorten der Wintersemester 2022/23 und 2023/24 erhoben, mittels einer Umfrage nach dem Grad der Forderung in der Veranstaltung „Softwareentwicklung 1“ drei bis vier Wochen nach Semesterbeginn. Abbildung 5 stellt die Ergebnisse gegenüber und

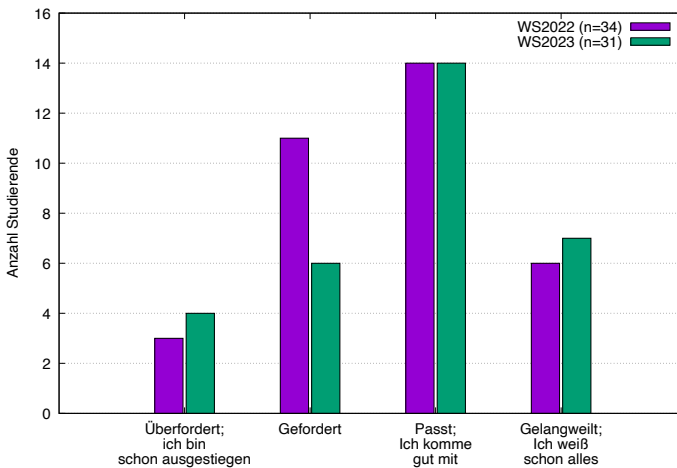


Abb. 5: Grad der gefühlten Herausforderung etwa drei bis vier Wochen nach Semesterbeginn

verdeutlicht deren große Ähnlichkeit. Lediglich knapp die Hälfte der Teilnehmenden sagt aus, dass das Tempo passt und sie gut mitkommen, etwa 20% der Studierenden langweilen sich und der Rest ist stark gefordert oder gar überfordert.

## 3 Mögliche Maßnahmen

Im Folgenden stellen wir einige Maßnahmen vor, die sich in unserer Lehrpraxis im Umgang mit den heterogenen Studierenden und insbesondere deren heterogenen Vorkenntnissen

bewährt haben. Grundlage unserer Lehre sind dabei explizit definierte, kompetenzorientierte Lernziele [Th15a] sowie das Prinzip des Constructive Alignments [BT11], gemäß dem wir die Prüfung abgestimmt mit den Lernzielen gestalten und Lehr-Lernmethoden konsequent auf das Erreichen dieser Ziele hin ausrichten.

### **3.1 Differenzierte Übungsblätter**

Je nach dem Detaillierungsgrad eines Übungsblatts bzw. einer Aufgabenstellung für eine Praktikumsaufgabe können Studierende überfordert oder gelangweilt werden. Wir experimentieren dabei bereits seit einiger Zeit damit, Übungsblätter in zwei Varianten zu erstellen und diese explizit den Studierenden zur Auswahl anzubieten [ZBT17]. Dabei adressiert eine Variante die Studierenden mit guten Vorkenntnissen in der Softwareentwicklung und die andere Variante die eher herausgeforderten Studierenden.

### **3.2 Homogenisierung von Gruppen**

Um gezielt auf die Bedarfe einer bestimmten Gruppe von Studierenden eingehen zu können kann es hilfreich sein, die Kohorte für ausgewählte Phasen zu homogenisieren.

#### **3.2.1 Erstsemester-Einstiegsprojekt**

Beim Erstsemester-Einstiegsprojekt durchlaufen die Studierenden anhand eines Beispiels und mit Hilfe von haptischen und visuellen Materialien die Kernaktivitäten des Software Life Cycles, also Anforderungsanalyse, Entwurf, Umsetzung und Test, in einem iterativ-inkrementellen Prozess [Th15b]. Da die Studierenden der Kohorte sich zu diesem Zeitpunkt gegenseitig so gut wie nicht kennen, entstehen bei freier Gruppeneinteilung nach dem Zufallsprinzip meist heterogene Gruppen, in denen Leute mit Vorkenntnissen auf Leute ohne Vorkenntnisse treffen.

Die Erfahrung zeigt, dass diejenigen Studierenden, die schon fachliche Vorkenntnisse haben (oder zu haben glauben) das Geschehen in diesen Gruppen stark dominieren. In der Folge halten sich Studierende ohne Vorkenntnisse mit ihren Ideen oft eher zurück und beobachten weit gehend passiv, was die (vermeintlich) vorgebildeten als Lösungsweg erarbeiten.

Hier haben wir gute Erfahrungen gemacht mit einer Homogenisierung nach Vorkenntnissen, auf der Grundlage einer studentischen Selbsteinschätzung. Dabei challengen sich die Studierenden in den Gruppen mit Vorkenntnissen untereinander und wirken dabei als gegenseitiges Korrektiv. Die Gruppen ohne Vorkenntnisse orientieren sich gemeinschaftlich, Schritt für Schritt und in untereinander ähnlichem Tempo an der Anleitung und diskutieren auf dem Weg bestehende Unklarheiten.

### 3.2.2 Zusatztutorial

Ein weiteres Beispiel für ein Lernsetting mit einer homogenisierten Gruppe ist das Zusatztutorial, das wir seit dem Wintersemester 2022/23 anbieten. Hierfür sprechen wir gezielt Studierende an und ermuntern sie zur Teilnahme. Grundlage dafür ist die in Abschnitt 2.4 vorgestellte Umfrage nach dem Grad der Forderung in der Veranstaltung drei bis vier Wochen nach Semesterbeginn (siehe Abbildung 5).

Dieses Zusatztutorial bieten wir gezielt für die Studierenden aus der Gruppe der Geforderten bzw. Überforderten an. Anderen Studierenden verwehren wir explizit den Zugang, um die Gruppe von der fachlichen Leistungsfähigkeit her möglichst homogen zu halten. Gleichzeitig schaffen wir so einen geschützten Raum, in dem sich alle trauen, ihre Fragen offen zu stellen, weil allen klar ist, dass die anderen auch nicht mehr wissen als man selbst.

Durch diese Vorgehensweise vermeiden wir außerdem den üblicher Weise zu beobachtenden Effekt, dass «exakt die „falschen“ Student\*innen zu Tutorien und zu Vorkursen kommen»<sup>6</sup> – also die, die es eigentlich gar nicht nötig haben.

Von den Studierenden, die wir über die Umfrage zur Klassifizierung des eigenen Gefordert-Seins gezielt angesprochen haben, hat im Wintersemester 2022/23 rund die Hälfte, im Wintersemester 2023/24 rund zwei Drittel der von Forderung oder Überforderung betroffenen Studierenden von unserem Angebot Gebrauch gemacht.

Das Tutorium wird von zwei Dozierenden im Modus des Pair-Teaching durchgeführt [ZBB18], um unterschiedliche Zugänge zu den Themen zu ermöglichen.

### 3.3 Heterogenisierte Gruppen

Heterogenisierte Gruppen sind insbesondere bei Projektarbeiten nützlich, wenn die einzelnen Beteiligten sich mit ihren individuellen Skill-Sets gegenseitig gut ergänzen. Hilfreich ist dabei allerdings ein gemeinsam zugrunde liegendes Wertesystem, um nicht permanent Grundsatzdiskussionen führen zu müssen, sowie eine offene Wertschätzung für eben genau die Diversität in der Gruppe.

Da in der Veranstaltung Softwareentwicklung 1 alle Erstsemester-Studierenden das Programmieren selbst erlernen sollen, fordern wir von den Studierenden individuelle Abgaben. Um den gegenseitigen Austausch auf dem Weg zur Lösung zu fördern, fordern wir von den Studierenden immer wieder gegenseitige Peer Reviews ihrer Lösungen ein, inklusive einer anschließenden kritischen Auseinandersetzung mit dem Feedback, das die Studierenden selbst zu ihrer eigenen Lösung erhalten haben.

---

<sup>6</sup> Siehe dazu beispielsweise diesen Blog-Eintrag von Jörn Loviscach: [https://j317h.de/blog/2023-05-17\\_09\\_44\\_VomVerschwindenderTutorien](https://j317h.de/blog/2023-05-17_09_44_VomVerschwindenderTutorien)

Als besonders Erkenntnis stiftend herauskristallisiert hat sich dabei das gezielte Bilden von Review-Tandems, die in ihren fachlichen Fähigkeiten möglichst heterogen sind. Die Schwächeren erhalten von den leistungsstärkeren Studierenden Hinweise zu möglichem Verbesserungspotenzial ihrer Lösung und sehen gleichzeitig aufgeräumte Lösungen mit einer gewissen Quelltext-Qualität. Umgekehrt erhalten die leistungsstärkeren Studierenden Hinweise zu Lücken in der Dokumentation bzw. zu Quelltext-Stellen, die schwer lesbar sind, und lernen gleichzeitig, sich in andere Lösungen und die dahinter liegenden Gedankengänge einzuarbeiten.

Dieser Ansatz lässt sich ausbauen bis hin zum kollegialen studentischen Coaching über einen längeren Zeitraum, beispielsweise die gesamte Studieneingangsphase.

### **3.4 Videos aufzeichnen**

Im letzten der Corona-Semester, also im Sommer 2022, konnten wir Erfahrungen mit hybrider Lehre bei gleichzeitiger Aufzeichnung der Lehrveranstaltung machen. Dabei hat überrascht, dass die Aufzeichnungen deutlich stärker genutzt wurden als erwartet [Bö23]. Es zeigt sich, dass insbesondere das Aufzeichnen von Aktivitäten in der Entwicklungsumgebung für die Studierenden sehr hilfreich ist, weil die einzelnen Bedienungsschritte oft so schnell ablaufen, dass die Studierenden diese nicht mental aufnehmen und verarbeiten können. Über die Aufzeichnung bekommen die Studierenden die Möglichkeit, die Bedienungsschritte in ihrem eigenen Tempo noch einmal nachzuvollziehen und ggf. am eigenen System mit durchzuführen.

## **4 Wirkung**

Der Nachweis der Wirksamkeit der Maßnahmen ist im Bildungsbereich meist schwer zu führen. So lässt sich nicht nachweisen, ob Studierende die Prüfung ohne eine Maßnahme wie einem zusätzlichen Tutorium nicht bestanden hätten.

### **4.1 Zusatztutorium**

Im Wintersemester 2022/23 haben von sieben regelmäßig Teilnehmenden des Zusatztutoriums vier die Modulabschlussprüfung im ersten und zwei im zweiten Versuch (Sommer 2023) bestanden. Zur Evaluation des Zusatztutoriums wurden von einer studentischen Hilfskraft aus dem Studiengang „Soziale Arbeit“ Interviews durchgeführt. Insbesondere wurden die Teilnehmenden gefragt, ob das Tutorium in dieser Form geholfen hat und weiter angeboten und entwickelt werden soll.

Folgende Rückmeldungen sind repräsentativ:

- «Raum für Fragen»
- «Es werden alle Fragen beantwortet»
- «Das Zusatztutorial, was der Herr Böttcher anbietet. Weil wir sind in einer recht kleinen Gruppe, ich glaube dieses Semester sind wir fünf Leute, und dann traut man sich eher Fragen zu stellen. Und man kann auch dumme Fragen stellen, weil man weiß halt, der andere würde auch vielleicht seine Fragen und – so alle auf demselben Level.»

## 4.2 Allgemeiner Lernfortschritt

Im Wintersemester 2022/23 haben wir die Studierenden am Ende des Semesters rückblickend ihre Startfähigkeiten relativ zu den für die Veranstaltung definierten Lernzielen einordnen lassen (siehe Abbildung 3), sowie zusätzlich ihren individuellen Lernfortschritt in den verschiedenen Themenbereichen. Abbildung 6 zeigt beispielhaft für die Themenbereich „Klassen bauen“, „Algorithmus finden“ und „Unit-Testen“, dass zu jedem Thema *alle* Studierenden aus der Veranstaltung etwas für sich mitgenommen haben, unabhängig von ihren Vorfähigkeiten.

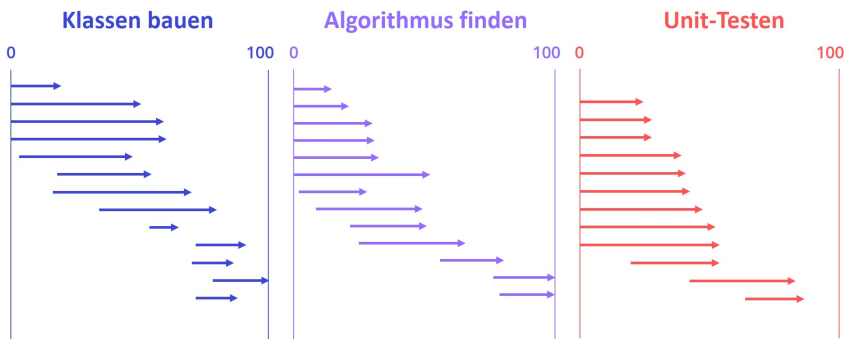


Abb. 6: Fachlicher Lernfortschritt differenziert nach Themenbereichen, in % der Lernzielerfüllung

Welche der durchgeführten Maßnahmen dabei wie stark zum Lernerfolg beigetragen haben lässt sich jedoch nicht identifizieren.

## 5 Fazit

In dieser Arbeit haben wir eine Reihe von Ansätzen und Maßnahmen vorgestellt, die sich in der Grundlagenausbildung zu Softwareentwicklung für das Lehren und Lernen von und mit heterogenen Kohorten in der informatischen Studieneingangsphase bewährt haben. Viele dieser Ansätze sind von ihrer jeweiligen Grundidee her auch in andere Lehr-Lernkontexte übertragbar.

## Literatur

- [An01] Anderson, L. W.; Krathwohl, D. R.; Airasian, P. W.; Cruikshank, K. A.; Mayer, R. E.; Pintrich, P. R.; Rath, J.; Wittrock, M. C.: A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. Longman, New York, 2001.
- [Be23] Bender, E.; Barbas, H.; Hamann, F.; Soll, M.; Sitzmann, D.: Fähigkeiten und Kenntnisse bei Studienanfänger\*innen in der Informatik: Was erwarten die Dozent\*innen? Hochschuldidaktik Informatik HDI 2021 (Commentarii informaticae didacticae)/13, S. 279–299, 2023.
- [Bö23] Böttcher, A.: Experiences with Low-to-Medium-Effort Hybrid Teaching. In: IEEE Global Engineering Education Conference (EDUCON). S. 1–7, 2023.
- [BT11] Biggs, J.; Tang, C.: Teaching For Quality Learning At University. McGraw-Hill Education, 2011, ISBN: 9780335242757.
- [De23] Desel, J.; Opel, S.; Siegeris, J. (Hrsg.): Hochschuldidaktik Informatik HDI 2021. In: Commentarii informaticae didacticae. 9. Fachtagung des GI-Fachbereichs Informatik und Ausbildung/Didaktik der Informatik 15.-16. September 2021 in Dortmund 13, 2023.
- [He48] Helwig, P.: Das Wertequadrat. *Psyche* 2/1, S. 121–127, 1948.
- [SK22] Soll, M.; Kobras, L.: What Were We Expecting? Analysing Expectations of German University Teachers of Study Beginners in Computer Science as Experienced by Students. In: 2022 IEEE German Education Conference (GeCon). S. 1–6, 2022.
- [TBH21] Thurner, V.; Böttcher, A.; Häfner, T.: A Detailed Analysis of Gender Differences in the Course of CS-Studies. In: 2021 IEEE Global Engineering Education Conference (EDUCON). S. 482–491, 2021.
- [Th15a] Thurner, V.; Böttcher, A.; Schlierkamp, K.; Zehetmeier, D.: Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen. In: Tagungsband des 14. Workshops Software Engineering im Unterricht der Hochschulen 2015, Dresden, Deutschland, 26. - 27. Februar 2015. S. 9–20, 2015.
- [Th15b] Thurner, V.; Schlierkamp, K.; Zehetmeier, D.; Böttcher, A.: Software Engineering Project Simulation in Student Entry Phase of Computer Scientists-to-be. In: Proceedings of the 2015 IEEE Global Engineering Education Conference (EDUCON), Tallinn. S. 486–493, März 2015.
- [ZBB18] Zehetmeier, D.; Böttcher, A.; Brüggemann-Klein, A.: Designing Lectures as a Team and Teaching in Pairs. In: Proc. 4th International Conference on Higher Education Advances (HEAd'18), Valencia. S. 873–880, 2018.
- [ZBT17] Zehetmeier, D.; Böttcher, A.; Thurner, V.: Differenzierte Übungsblätter – Für Experten und solche die es werden wollen. In: Tagungsband zum 3. Symposium zur Hochschullehre in den MINT-Fächern. S. 94–98, Sep. 2017.

# Ein kombiniertes Rahmenmodell zum Programmierenlernen

## Entwicklung, Erprobung und Evaluation einer Lehrveranstaltung für das Programmierenlernen unter Benutzung von Jupyter Notebook

Robert Ringel<sup>1</sup> und Hermann Kördle<sup>2</sup>

**Abstract:** Der Beitrag zeigt das Konzept eines Rahmenmodells zum Programmierenlernen, welches auf Grundlage anerkannter lernpsychologischer Methodiken entwickelt wurde. Im Kern steht die Bearbeitung von Aufgabenfolgen gemäß der 4C/ID-Methodik. Sie wird von Prozessen der Reflexion und Erkenntnissicherung begleitet. Nach diesem Rahmenmodell wurde ein Grundlagenkurs zur Programmierausbildung an der HTW Dresden durchgeführt. Ausgewählte quantitative Ergebnisse und qualitative Aussagen von Studierenden zeigen Erfolge und Möglichkeiten der Weiterentwicklung des vorgeschlagenen Rahmenmodells.

**Keywords:** Programmierenlernen, Aufgabenfolgen, 4C/ID, Rahmenmodell, Methodik

## 1 Einleitung

Das Programmierenlernen ist seit vielen Jahren fester Bestandteil der Ingenieurs- und Informatikausbildung an Hochschulen. Das Lernziel besteht in der Befähigung, grundlegende ingenieurtechnische Probleme mit Hilfe von Computerprogrammen zu lösen. Seit den 1980er Jahren erfolgt eine wachsende Erforschung der Lehr-Lern-Prozesse in der Informatik, die sich insbesondere auf den Bereich des Programmierenlernens fokussiert. Robins [Ro19] hat für den Zeitraum 1980-2018 über 200 Quellen aus diesem Bereich analysiert. Im Handbuchkapitel mit dem Titel „Novice programmers and introductory programming“ stellt er die Charakteristika des Programmierens, die Probleme der Lernenden, Aussagen zu Lernergebnissen sowie generelle Hinweise zum Lehren und Lernen des Programmierens anhand der wissenschaftlichen Befundlage zusammen. Bedeutsam ist dabei seine Feststellung, dass es unter den Lehrenden kein einheitliches Verständnis zum elementaren methodischen Vorgehen in Grundlagenkursen des Programmierens gibt. Robins geht noch weiter, in dem er feststellt, dass nicht einmal ein Konsens über die wesentlichen Fachinhalte und deren begründete Sequenzierung im Lernprozess besteht.

Dieser Beitrag skizziert die Konzeption eines Rahmenmodells zum Programmierenlernen, welches auf der Grundlage anerkannter lernpsychologischer Modelle entwickelt wurde. Der Beitrag zeigt die daraus resultierende Implementierung einer Lehrveranstaltung zum Erlernen der Python-Programmierung. Praktische Erfahrungen, die Darstellung von

---

1 HTW Dresden, Fakultät Informatik/Mathematik, PF 120 701, 01008 Dresden, Deutschland, robert.ringel@htw-dresden.de

2 TU Dresden, Fakultät Psychologie, 01062 Dresden, Deutschland, hermann.koerdle@tu-dresden.de

Prüfungsleistungen sowie Aussagen von Studierenden belegen das Potential des Rahmenmodells.

## **2 Entwurf eines lernpsychologisch fundierten Rahmenmodells zum Programmierenlernen an Hochschulen**

Aebli [Ae98] beschreibt in seinem Standardwerk „Zwölf Grundformen des Lehrens“ den vollständigen Lernzyklus. Er besteht aus den Phasen: Aufbauen, Durcharbeiten, Üben und Wiederholen sowie Anwenden. Darauf aufbauend schlagen Wespi, Luthiger, Wilhelm [WLW15] ein allgemeines Vorgehensmodell zur Gestaltung kompetenzorientierter Aufgabensets vor, welches aufgabenbasiertes Lernen in eben diesen vollständigen Lernzyklen ermöglicht.

Van Merriënboer und Kirschner [MK18] entwickeln mit Four-Component Instructional Design (4C/ID) eine aufgabenbasierte Methodik, die eine konkrete Umsetzung zur Gestaltung von Aufgabensets nach [WLW15] darstellt. Das Ziel von 4C/ID besteht darin, komplexes Lernen zu ermöglichen und einen umfassenden, zusammenhängenden Aufbau von Wissen, Handlungsfähigkeit und Problemlösekompetenz zu gewährleisten. Dabei kommen ganzheitliche Aufgabenstellungen mit fachpraktischem Hintergrund zum Einsatz. Im Kern der Methodik stehen Folgen von Lernaufgaben, die mit unterstützender Information und Anleitungen zu Handlungsabläufen untersetzt sind. Die Aufgabenanordnung erfolgt so, dass eine nachlassende aufgabeninhärente Unterstützung den Anforderungsgrad systematisch steigert. Gleichzeitig sorgt die inhaltliche Variabilität der Aufgabenschwerpunkte dafür, die Herausbildung und Festigung von Konzeptwissen und Problemlösestrategien zu ermöglichen. Für das Programmierenlernen schlägt van Merriënboer [Va90] vor, zunächst Aufgaben zum Programmverstehen und zum Nachprogrammieren bekannter Lösungen zu bearbeiten. Erst gegen Ende des Lernzyklus sollen Programme allein auf Grundlage der Aufgabenstellung vollständig neu geschrieben werden.

Task-Based Learning (TBL) nach Willis [Wi04] ist eine aufgabengestützte Methodik aus dem Bereich des Fremdsprachenlernens. TBL benutzt jedoch nach dem Zyklus der Aufgabenbearbeitung einen dedizierten Lernschritt zur Erkenntnissicherung und Fokussierung auf sprachliche Korrektheit. Im letzten Lernschritt erfolgt nochmals reale sprachliche Anwendung. Damit hebt TBL das Defizit einer fehlenden Konsolidierung oder Erkenntnissicherung von 4C/ID auf.

Die Kombination beider Methodiken führt zu einem kombinierten Rahmenmodell gemäß Abbildung 1. Dabei bildet die TBL-Methodik mit den Phasen Pre-Task, Task, Post-Task und der abschließenden Complex-Task den Rahmen, in welchen die Aufgabenfolgen gemäß der 4C/ID-Methodik eingebettet sind.

Vor Beginn der Arbeit an den Aufgabenfolgen sollten die Lernenden durch ein Video oder durch einen fachlichen Vortrag grundlegendes Wissen zum Lernbereich erworben haben. In



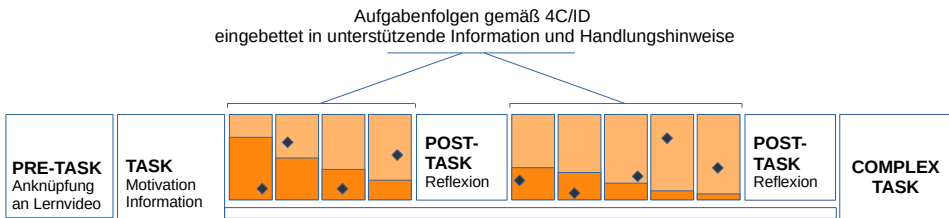


Abb. 1: Schematische Darstellung des kombinierten Rahmenmodells

einer Pre-Task-Phase zu Beginn der Aufgabenfolge wird der Anschluss daran hergestellt, indem wesentliche Inhalte des Videos rekapituliert werden. Dazu führt die Lehrkraft im Auditorium ein kurzes Live-Coding mit den neuen Programmiererelementen durch. In der Task-Phase wird die fachliche Motivation für den Themenbereich aufgebaut, indem die Lernenden überlegen, welche Möglichkeiten für die fachliche Anwendung aus den neuen Programmiermöglichkeiten erwachsen. Entsprechende Erkenntnisse werden schriftlich im Arbeitsmaterial festgehalten. Der Informationsteil der Task-Phase schafft den Zugang zu weiteren Informationsquellen des jeweiligen Fachthemas. Dies können Verweise auf Buchkapitel oder entsprechende Webseiten sein. Diese unterstützende Information steht während der folgenden Aufgabenbearbeitung zur Verfügung. Danach beginnt die Bearbeitung der Aufgabenfolge mit variablen inhaltlichen Schwerpunkten und einer rückläufigen aufgabeninhärenten Unterstützung in Form von vorgegebenem Quelltext oder integrierten Code-Kommentaren. Wichtig ist nach einer Anzahl von 3-5 Lernaufgaben im Bereich Post-Task eine Reflexion zu den Aufgabenergebnissen und ggf. eine explizite schriftliche Erkenntnissicherung vorzunehmen. Am Ende der Aufgabenfolge eines Themenbereichs steht eine Komplexaufgabe. Damit wenden die Studierenden das Gelernte in einer umfangreichen, realen Fachaufgabe problemlösend an. Dieser Zyklus einer Aufgabenfolge ist für jeden Lernbereich des Programmierlehrgangs zu durchlaufen.

Wichtig für den Erfolg der Methodik ist eine große Anzahl Lernaufgaben mit ausreichender Variabilität der Aufgabenstellung. Dem gegenüber steht der klassische Lehrbetrieb an Hochschulen. Eine Woche besteht in der Regel aus einem Vorlesungsanteil und einem Übungsanteil. Die eigene Lehrerfahrung in der Informatik zeigt, dass eine praktische Übungszeit von 90 Minuten pro Woche für die Mehrzahl der Studierenden nicht ausreicht, um zu den Vorlesungsinhalten gefestigte praktische Fähigkeiten aufzubauen. Daher empfiehlt es sich, das vorgeschlagene Rahmenmodell im Lehrbetrieb so zu realisieren, dass pro Woche mindestens zweimal 90 Minuten aktives Lernen an den Aufgabenfolgen stattfinden kann. Dazu werden die Vorlesungsinhalte in Themenvideos verfügbar gemacht. Die Videos im zeitlichen Umfang von 15-20 Minuten müssen die Lernenden vor Beginn einer Aufgabenfolge angeschaut haben, um die fachliche Einführung in das Themengebiet zu erhalten. Die Arbeit an der Aufgabenfolge beginnt dann mit einer kurzen Erinnerung an Kerninhalte des Videos. Durch den Wegfall der Vorlesung zugunsten einer weiteren Übungsstunde haben die Lernenden innerhalb einer Woche an drei Tagen die Möglichkeit, sich mit

dem Programmierenlernen zu beschäftigen. Dabei wirkt regelmäßige Programmierarbeit, idealerweise im Rhythmus von 2-3 Tagen, dem natürlichen Vergessenprozess systematisch entgegen.

### 3 Gestaltung der Aufgabenfolgen

Eine Grundlage für die Entwicklung von Aufgabenfolgen einer Lehrveranstaltung bilden die Lernziele. Van Merriënboer und Kirschner schlagen vor, die Lernziele in einer Hierarchie anzuordnen [MK18, S. 14]. Diese Hierarchie ist der Ausgangspunkt für die nachfolgende Sequenzierung der Ziele. Daher ist die Leserichtung des Baumes auf der Blattebene von links nach rechts definiert. Lernziele, deren Erarbeitung parallel integriert erfolgt, sind durch Doppelpfeile zu kennzeichnen. Die Abbildungen 2 und 3 zeigen die Lernzielhierarchie für den Kurs zu den Grundlagen der Pythonprogrammierung. Durch die Doppelpfeile an den Knoten „Beherrschung der Programmiersprache“ und „Entwicklung von Problemlösungen“ wird gezeigt, dass das Problemlösen integrierter Bestandteil der Beherrschung der Programmiersprache ist. Das Qualifikationsziel dieses Kurses besteht in der problemlösenden Anwendung der Programmiersprache Python.

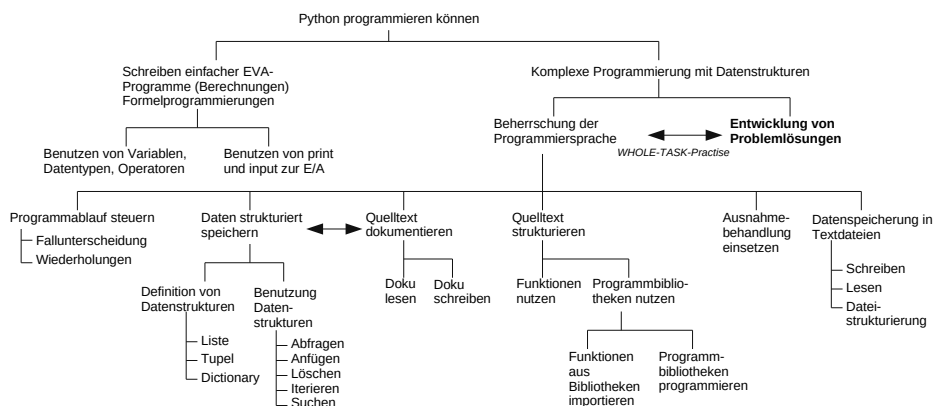


Abb. 2: Beherrschung der Programmiersprache als Teil 1 der Lernzielhierarchie des Grundlagenkurses zum Python-Programmieren

Aus der hierarchischen Darstellung wird nun der Lehrveranstaltungsplan für die jeweilige Zahl der Semesterwochenstunden abgeleitet und beispielsweise in Tabellenform festgehalten. In der Tabelle sollten die Lernziele in den Dimensionen Wissen, Handlungsfähigkeit, zu lösende Probleme untersetzt werden, um so, in Anlehnung an die Kompetenzdefinition der Arbeitsgruppe „Computing Curricula 2020“ [CC21, S. 126], eine zielgerichtete Kompetenzentwicklung zu erreichen. Diese Informationen sind zugleich wesentliche Inhaltsangaben zur Gestaltung der Lernaufgaben. Außerdem lässt sich anhand des Plans ableiten, welche fachlichen Themen als Lehrvorträge oder Lehrvideos zu welchen Zeitpunkten erforderlich sind, um die notwendigen Grundkenntnisse für die Bearbeitung der Programmieraufgaben

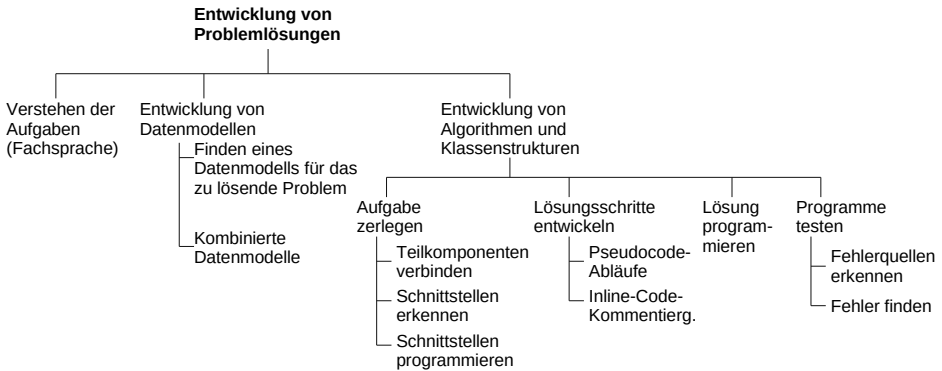


Abb. 3: Entwicklung von Problemlösungen als Teil 2 der Lernzielhierarchie des Grundlagenkurses zum Python-Programmieren

bereitzustellen. Zudem soll die Tabelle zeigen, welche zeitlichen Umfänge für die jeweiligen Aufgabenfolgen zur Verfügung stehen. Damit ist der formale Rahmen dargestellt, der durch die Folgen von Lernaufgaben zu füllen ist.

Im nächsten Arbeitsschritt wird für jeden Lernbereich gemäß der Tabelle ein Pool von Lernaufgaben entwickelt. Dabei besteht das Ziel darin, solche Aufgaben zu gestalten, die das Fachwissen festigen, Konzeptverständnis fördern und notwendige Handlungssicherheit entstehen lassen [KP23]. Dies bildet die Grundlage, um mit Hilfe komplexer Aufgabenstellungen fachliche Probleme im jeweiligen Themenbereich zu bearbeiten. **Dabei sind die Handlungen der Lernenden im Zuge der Aufgabenbearbeitung das entscheidende Element für die Gestaltung lernwirksamer Aufgabenstellungen.**

Tabelle 1 zeigt einen Aufgabenpool aus dem Abschnitt Datenstrukturen des Grundlagenkurses zur Python-Programmierung. Das Lernziel dieses Kursabschnitts besteht im problemlösenden Anwenden der Datenstrukturen Liste, Tupel und Dictionary. Die Tabelle macht deutlich, wie die Lernziele, Lernhandlungen und die Aufgabentypen der einzelnen Aufgaben aufeinander abgestimmt sind. Insgesamt sind für den Themenbereich acht Aufgaben ansteigender Schwierigkeit zur Bearbeitung verfügbar, bevor am Ende eine Komplexaufgabe bearbeitet wird. Im Ergebnis der Aufgabenlösung dieser Komplexaufgabe wird die erreichte Kompetenz der Lernenden im Kursabschnitt sichtbar. Die Darstellung und Ausarbeitung des zugrundeliegenden Kompetenzmodells nach [CC21, S.126] geht über den Rahmen dieses Beitrags hinaus. Sie ist jedoch ein wertvolles Instrument zur Beurteilung der erreichten Lernziele.

Für die Implementierung der Aufgabenfolgen gemäß dem kombinierten Rahmenmodell eignen sich auch digitale Lehr-Lern-Umgebungen. Dies sollte insbesondere bei Grundlagenkursen zum Programmierenlernen bedacht werden [SB23]. Die Aufgabeninhalte und die Aufgabenumfänge der Programmieraufgaben von Grundlagenkursen machen es nicht

Aufgabe	Lernziel	Lernhandlungen	Aufgabentyp	Anmerkung
A	Listen ersetzen Skalarvariablen	lesen, gedanklich verändern, Notizen aufschreiben	Worked- Out- Example	Partner- gespräch
B	Programmierung Liste	Teile implementieren, testen	Comple- tion	-
C	Inhomogene Listen	Code lesen, vergleichen, ausführen, Erkenntnis notieren	Worked- Out- Example	Partner- gespräch
D	Problemlösung mit Listen	Datenstruktur skizzieren, Lösungsansatz finden, Codestruktur entwerfen, Code implementieren, testen	Conven- tional Task	Muster- lösung, Reflexion: Problem- to-Code
E	Zusammengesetz- ter Key für Dictionary; Dictionary speichert Liste	lesen, Kurzkommentare schreiben, Programm- funktion beschreiben	Worked- Out- Example	Partner- gespräch
F	Programmierung Dictionary	lesen, gedanklich verändern, implementieren, testen	Comple- tion	Muster- lösung
G	Programmierung Set	Begriff erarbeiten, Begriff transferieren, Code implementieren	Comple- tion	Lösung für alle darstellen
H	Grundoperationen aller Datenstrukturen	lesen, Details erkennen, markieren, vergleichen, klassifizieren	Worked- Out- Example	Lösung für alle darstellen
Complex Task	Problemlösen mit Datenstrukturen	Datenstruktur skizzieren, Problemlösung entwickeln, Codestruktur entwerfen, implementieren, testen	Conven- tional Task	Gestuftes Hilfe- system, Lösung für alle darstellen

Tab. 1: Übersicht zum Aufgabenpool des Themenbereichs der Datenstrukturen

erforderlich, professionelle Entwicklungswerkzeuge einzusetzen. Mit Jupyter Notebook steht eine einfache Programmierumgebung zur Verfügung, welche nach [Re20], [Se20] und [Za19] erfolgreich für die Lehre einsetzbar ist. Ein Vorteil von Jupyter Notebook besteht darin, dass Aufgabentexte, ergänzende Informationen und grafische Darstellungen gemeinsam in einem Medium mit dem ausführbaren Programmquelltext integriert sind. Dabei können alle Arten von Informationen durch die Lernenden genauso bearbeitet werden wie der Programmcode. Damit ist Jupyter Notebook ein wirklich interaktives digitales Lernmedium. Die grundlegenden Arbeitsmaterialien werden durch die Lehrkraft gestaltet und im Kurs publiziert. Sie bilden dabei unmittelbar das Konzept des kombinierten Rahmenmodells zum Programmierenlernen gemäß Abbildung 1 ab. Im Zuge der Aufgabenbearbeitung tragen die Lernenden ihre Arbeitsergebnisse in diese Materialien ein. Insbesondere die Möglichkeit zur individuellen Reflexion und zur schriftlichen Erkenntnissicherung machen Jupyter Notebook zu einem wertvollen Lernwerkzeug, da es für die Lernenden hilfreich ist, neben der reinen Aufgabenlösung auch Erkenntnisse festzuhalten. Die Art der Erkenntnisgewinnung kann als eine Reflexion von Lösungsvarianten im Partnergespräch stattfinden. Das Beschreiben der eigenen Lösungen bzw. das Verstehen einer alternativen Lösung stellen dabei wertvolle Lernhandlungen dar. Diese Handlungen sind wichtige Ergänzungen, die über den unmittelbaren Problemlösungsprozess hinausgehen.

## 4 Ergebnisse und praktische Erfahrungen

Der folgende Abschnitt zeigt Ergebnisse und praktische Erfahrungen bei Durchführung der Lehrveranstaltung „Grundlagen der Python-Programmierung“ im Lehrbetrieb der Hochschule. Ausgehend von den organisatorischen Rahmenbedingungen des Lehrbetriebs werden die Ergebnisse quantitativer und qualitativer Evaluierungen dargestellt. Das Kapitel endet mit einer kritischen Reflexion zu den Erfahrungen im Lehrbetrieb.

### 4.1 Erfahrungen aus dem praktischen Lehrbetrieb der Hochschule

Der Python-Lehrgang wurde nach dem oben dargestellten methodischen Konzept des kombinierten Rahmenmodells (s. Abb. 1) gestaltet und mit Hilfe von Jupyter-Notebook implementiert. Er wurde in zwei Jahrgängen des Bachelor-Studiengangs Wirtschaftsingenieurwesen im 1. Semester als Pflichtmodul für jeweils 70-80 Studierende im Wochenumfang von zwei Doppelstunden an der Hochschule für Technik und Wirtschaft Dresden durchgeführt. Als Voraussetzung mussten alle Studentinnen und Studenten einen eigenen Notebook-Computer mit einer Jupyter-Notebook-Installation verfügbar haben. Anders als zunächst angenommen war dies kein Problem. Innerhalb einer Woche waren alle Studierenden damit ausgestattet und arbeitsfähig. Auf dieser Grundlage konnten die Lehrveranstaltungen gemäß dem Stundenplan im Umfang von zweimal 90 Minuten pro Woche sowohl im Hörsaal als auch in Seminarraum jeweils als praktische Programmierübungen durchgeführt werden. Für den Hörsaal bedeutet dies, eine solche Sitzordnung einzurichten, dass die Lehrkräfte bei

individuellen Fragen und Problemen, die einzelnen Studierenden erreichen können. Auch dies war alsbald gängige Praxis im Hörsaal.

In beiden Kursdurchgängen zeigte sich nach einer kurzen Eingewöhnungsphase eine aktive, von gegenseitigem Vertrauen geprägte Lernatmosphäre. Das Bestreben, in diesem neuartigen, auf die aktive Tätigkeit der Lernenden gerichteten Format, praktisch programmieren zu lernen, war bei der überwiegenden Mehrzahl der Studierenden deutlich erkennbar. Die Anregung, sich bei Fragen und Problemen gegenseitig zu helfen und Aufgabenlösungen mit dem Sitznachbarn zu besprechen wurde gern angenommen. Diese sozialen Interaktionen wurden fester Bestandteil der Arbeitskultur des Kurses.

Gemäß Moduldefinition der Hochschule muss diese Lehrveranstaltung mit einer schriftlichen Prüfung abschließen. Da es zum Zeitpunkt des Kurses nicht möglich war, für 80 Personen eine isolierte, digitale Prüfungsumgebung mit Jupyter Notebook aufzusetzen, musste die Prüfung mit Papier und Stift geschrieben werden. Dies war ein erheblicher Nachteil, da die Lernenden durch Jupyter Notebook natürlich ein echtes interaktives Programmieren inklusive Syntax-Hervorhebung und Testmöglichkeit für den Quelltext gewohnt waren. Um diesen Nachteil etwas auszugleichen, wurden in beiden Jahrgängen im Laufe des Semesters drei Lernstandserhebungen in schriftlicher Form durchgeführt. Hier hatten die Studierenden die Möglichkeit, sich in Bezug auf Inhalt, Anforderungsniveau und Art der Aufgabenstellungen für die Abschlussprüfung vorzubereiten. Zudem gaben die Lernstandserhebungen sowohl den Studierenden als auch der Lehrkraft eine zuverlässige Rückmeldung zur Erreichung der Lernziele. Die Lernstandserhebungen hatten den Umfang einer Doppelstunde. Die nachfolgende Lehrveranstaltung wurde dazu genutzt, dass die Lehrkraft die Lösung jeder Testaufgabe inklusive einer Anleitung zur Punktergabe vorstellte. Damit war es möglich, dass alle Lernenden ihre eigene Lösung möglichst objektiv nach einem vordefinierten Punktschema bewerten konnten. Die Punktergebnisse der Aufgaben wurden über ein digitales Formular erfasst und ergaben so ein Gesamtbild des Lernstandes im Kurs, welches für die Lehrkraft und die Lernenden einsehbar war.

Diese Art der Lernstandserhebung als Feedback-Instrument für die Lernenden und Lehrenden und gleichzeitig als methodisches Mittel zur systematischen Klausurvorbereitung, wurde von den Lernenden als wertvoll und zweckmäßig empfunden. Neben dem Test des eigenen Könnens verwiesen die Studierenden dabei auf die Lernwirkung der nachfolgenden intensiven Lösungsbesprechung und der damit verbundenen Transparenz bei der Punktergabe.

Die Prüfung selbst diagnostizierte Lernergebnisse in den unterschiedlichen Teilkompetenzen des Programmierens. Konzeptionelles Verständnis wurde anhand von Begriffsdefinition, Begriffsnetzwerken und der Zuordnung entsprechender Python-Anweisungen erfasst. Die Fähigkeiten im Problemlösen, in Handlungsfähigkeit und Transferleistungen waren in verschiedenen Teilaspekten auf die einzelnen Aufgaben verteilt. Insgesamt waren in 120 Minuten vier fachliche Aufgabenstellungen mit insgesamt zehn Teilaufgaben anhand vorgegebener Python-Quelltexte zu bearbeiten. Hierbei war überwiegend vorgegebener Pro-

grammquelltext zu dokumentieren und zu vervollständigen. Zudem waren Beispielskizzen der verwendeten Datenstrukturen anzufertigen. Der Anteil vollständig neu zu schreibenden Quelltexts belief sich auf wenige Funktionen im Umfang von 8-12 Codezeilen.

## 4.2 Ergebnisse der Evaluierungen in zwei Studienjahrgängen

Abbildung 4 zeigt die Boxplots der Prüfungsergebnisse beider Lehrveranstaltungsdurchgänge. Der Punktwert des Klausurergebnisses im Intervall 0-1 repräsentiert den Anteil erreichter Punkte bezogen auf die mögliche Maximalpunktzahl der Prüfung.

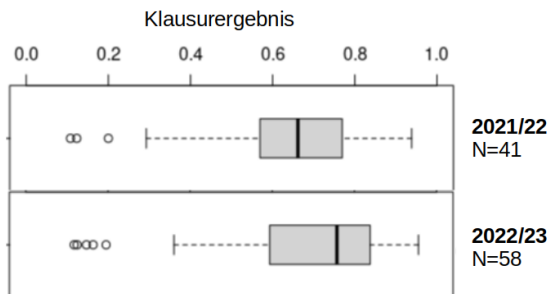


Abb. 4: Boxplots der Prüfungsergebnisse für zwei Lehrveranstaltungsdurchgänge

Für den Durchgang 2022/23 zeigt sich insgesamt ein besseres Prüfungsergebnis. Dies wird durch den höheren Medianwert von 0.75 und durch die höher liegenden Grenzen der Q1- und Q3-Quartile im Boxplot sichtbar.

Damit eine Möglichkeit eröffnet wird, das Lernergebnis des Kurses in Bezug zum Vorwissen der Studierenden darzustellen, wurde in der ersten Lehrveranstaltung auf freiwilliger Basis der Stand der Vorkenntnisse erfasst. Hierfür wurde den Kursteilnehmern eine Liste mit 30 Fachbegriffen des Programmierens vorgelegt. In dieser Liste waren jene Begriffe zu markieren, die so gut bekannt sind, dass die Überzeugung besteht, den jeweiligen Fachbegriff einer anderen Person erklären zu können.

Abbildung 5 zeigt zwei Quadranten-Plots, in denen das begriffliche Vorwissen in Bezug zum Klausurergebnis dargestellt wird. Von besonderem Interesse sind die beiden unteren Quadranten rechts (A) und mittig (B). In ihnen sind die Datenpunkte für jene Studierenden aufgetragen, die geringe bis mittlere begriffliche Vorkenntnisse hatten und die den Programmierkurs mit einem mittleren bis guten Ergebnis abschließen konnten. Es handelt sich also um jenen Teil der Kursgruppe, der ein positives Lernergebnis erzielt hat. Im unteren rechten Quadranten (A) befindet sich der Anteil von Personen mit einem guten Lernergebnis. Für das WS 2021/22 sind es 15% und für das Folgejahr 43%. In den unteren mittleren Quadranten (B) fallen die Indikationen einer mittleren Leistung. Es sind dies 44% für das WS2021/22 und 32% für das Folgejahr.

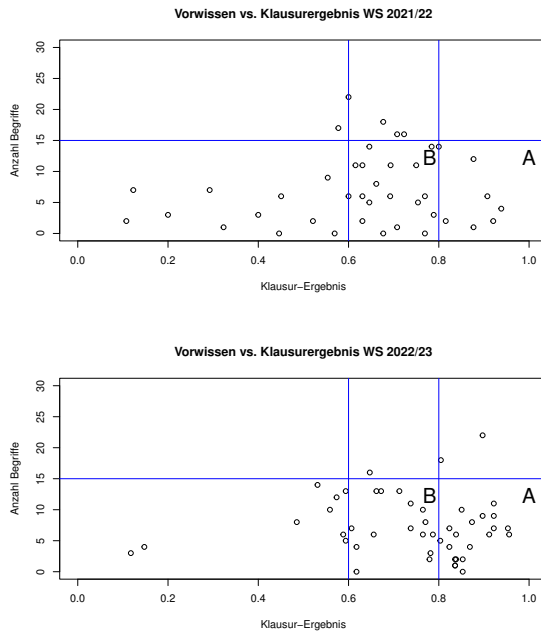


Abb. 5: Quadrantencharts der Prüfungsergebnisse in Bezug zum Vorwissen

Ergänzend zu den quantitativen Befunden anhand der Prüfungsleistungen wurden mit zufällig ausgewählten Kursteilnehmern Interviews als Leitfadengespräche in Anlehnung an [DBP16, S.372-373] durchgeführt. Neben einer Vielzahl individueller Details Aussagen finden sich wiederkehrende Äußerungen, deren Grundaussage in vielen Gesprächen getroffen wird. Einige Beispiele sollen an dieser Stelle aufgeführt werden.

Die Idee, durch den Einsatz von Lernvideos eine Möglichkeit zu schaffen, pro Woche zweimal aktiv zu programmieren, wird durchweg positiv aufgenommen. Ebenso wird der Einsatz von Jupyter Notebook als interaktives Lernwerkzeug für die Lehrveranstaltung und für das individuelle Arbeiten zu Hause sehr begrüßt. Im Wesentlichen wird Jupyter Notebook für das aktive Programmieren genutzt. Deutlich seltener machen die Studierenden ohne Aufforderung darin individuelle Eintragungen außerhalb der Lernaufgaben.

Die Lernstandserhebungen werden ebenfalls durchgängig begrüßt. Sie stellen nach Meinung der befragten Personen eine gute Möglichkeit der Überprüfung des eigenen Lernstands dar. In diesem Zusammenhang werden auch die Darstellung der Aufgabenlösung und die Transparenz zur Punktvergabe als hilfreich benannt.

Der Inhalt und die Gestaltung der Lernvideos werden überwiegend positiv bewertet. Dabei wird wiederholt angegeben, dass man sie im Gegensatz zu einer Vorlesung mehrfach anschauen kann, dass man sie unterbrechen, den Inhalt selbst am Computer nachvollziehen



kann und danach den Film fortsetzen kann. Einzelne Personen geben an, dass einige Videos zu lang oder zu „trocken“ sind.

Methodische Elemente, die mehrfach positiv als wichtiger Bestandteil des Lehrkonzepts benannt werden, sind: das Live-Coding zum Einstieg in die Aufgabenfolge, die Vielzahl der Übungsaufgaben und das gestufte Hilfesystem in den Jupyter Notebooks sowie die Partnerarbeit. Das Anforderungsniveau der Lehrveranstaltung wird von den meisten Befragten als angemessen bis anspruchsvoll eingeschätzt.

### **4.3 Kritische Reflexion der Erfahrungen und Ergebnisse**

Kritische Erfahrungen aus den beiden Durchgängen der Lehrveranstaltung betreffen die folgenden Bereiche: Es war zu beobachten, dass ein bestimmter Anteil der Lernenden wiederholt in die Lehrveranstaltung kam, ohne das Lernvideo angeschaut zu haben. Dabei zeigte sich, dass es für diese Personen deutlich schwieriger war, die Aufgabenfolgen zu bearbeiten, da natürlich das einführende Programmierbeispiel allein das Lernvideo nicht kompensieren kann.

Die reine Bearbeitung der Aufgabenfolgen reicht für viele Studierende nicht aus, um nachhaltige Erkenntnisse aufzubauen. Daher ist der Praxis einer regelmäßigen Ergebnisbesprechung und Lösungsdiskussion eine große Bedeutung beizumessen. Unterbleibt dies, so wurde wiederholt bemerkt, dass die Lernenden bei der Bearbeitung der Aufgabe 4 nicht mehr wissen, welche Erkenntnis aus Aufgabe 2 entstanden ist. Daher wurden im zweiten Durchgang der Lehrveranstaltung in den Jupyter Notebooks extra farblich hervorgehobene Bereiche zur Erkenntnissicherung ergänzt. An diesen Stellen wurde während der Praktika grundlegendes Konzeptwissen in Form von einfachen Merksätzen formuliert.

Im Rahmen des Problemlöseprozesses gehen die Studierenden häufig viel zu schnell zum Programmieren über. Die Aufforderung zum Anfertigen von Skizzen zu den Datenstrukturen als Basis der Problemlösung steht explizit im Aufgabentext. Dennoch wird sie immer wieder einfach übergangen. Zukünftig wäre es interessant, solche Aufgaben zu konstruieren, bei denen lediglich die Schritte der Problemanalyse und der Lösungskonstruktion verlangt werden, nicht aber die Formulierung des Programmcodes.

Generell fällt den Studienanfängern das Denken in Konzepten und Strategien schwer. Studierende mit einem Vorstudium oder mit einem beruflichen Hintergrund haben augenscheinlich weniger Schwierigkeiten bei diesen Anforderungen.

Obwohl die Lehrveranstaltung insgesamt überwiegend positive Lernergebnisse geliefert hat und die Mehrzahl der Studierenden am Ende über wesentliche Grundkenntnisse im Programmieren verfügt, bleibt ein erhebliches Risiko für den nachhaltigen Erfolg. Damit die erworbenen Kenntnisse und Fähigkeiten nicht verblassen, ist es unbedingt erforderlich, in den nachfolgenden Semestern in anderen Lehrveranstaltungen die Programmierfähigkeit systematisch einzufordern und zu praktizieren.

Positiv bleibt zu vermerken, dass wesentliche Ideen aus dem Design dieses Kurses, aus den Ideen der Aufgabengestaltung und zur Nutzung von Jupyter Notebook durch Kollegen aufgegriffen wurden, wodurch sie positiven Einfluss auf die Gestaltung weiterer Lehrveranstaltungen nehmen.

## 5 Zusammenfassung

Das kombinierte Rahmenmodell zum Programmierenlernen adaptiert lernpsychologisch fundierte Methodiken des aufgabenbasierten Lernens. Seine Lernwirksamkeit wurde in Grundlagenkursen der Python-Programmierung erfolgreich evaluiert. Dabei hat sich gezeigt, dass aufgabenbasiertes Lernen ohne Vorlesung erfolgreich durchführbar ist und dass diese Methodik von den Lernenden gern angenommen wird.

Jupyter Notebook ist bestens geeignet, um interaktive Lernmaterialien für Grundlagenkurse des Programmierenlernens zu erstellen. Es unterstützt neben den unmittelbaren Handlungen des Programmierens auch Lernhandlungen der individuellen Reflexion und Erkenntnissicherung. Diese wichtigen Lernhandlungen sind als fester Bestandteil in die Methodik des kombinierten Rahmenmodells integriert und in den Aufgabenfolgen verankert.

## Literatur

- [Ae98] Aebli, H.: Zwölf Grundformen des Lehrens eine allgemeine Didaktik auf psychologischer Grundlage; Medien und Inhalte didaktischer Kommunikation; der Lernzyklus. Klett-Cotta, Stuttgart, 1998.
- [CC21] CC-2020 Task Force: Computing Curricula 2020: Paradigms for Global Computing Education. Association for Computing Machinery, New York, NY, USA, 2021.
- [DBP16] Döring, N.; Bortz, J.; Pöschl-Günther, S.: Forschungsmethoden und Evaluation in den Sozial- und Humanwissenschaften. Springer, Berlin, 2016.
- [KP23] Körndle, H.; Proske, A.: Arbeitsplatznahe Lernaufgaben: Ihre Modellierung, Konstruktion und Einsatz in digitalen Lehr-Lernszenarien beruflichen Lernens. bwp@ Profil 8: Netzwerke – Strukturen von Wissen, Akteuren und Prozessen in der beruflichen Bildung. Digitale Festschrift für Bärbel Fürstenau zum 60. Geburtstag/, 2023, URL: [https://www.bwpat.de/profil-8\\_fuerstenau/koerndle-proske](https://www.bwpat.de/profil-8_fuerstenau/koerndle-proske).
- [MK18] Merriënboer, J. J. G. v.; Kirschner, P. A.: Ten steps to complex learning a systematic approach to four-component instructional design. Routledge, Taylor & Francis Group, New York, 2018.
- [Re20] Reades, J.: Teaching on Jupyter. REGION 7/, S. 21–34, 2020.

- [Ro19] Robins, A. V.: Novice Programmers and Introductory Programming. In: The Cambridge Handbook of Computing Education Research. Cambridge Handbooks in Psychology, Cambridge University Press, S. 327–376, 2019.
- [SB23] Schmolitzky, A.; Burau, H.: OPPSEE - Eine Online-Plattform zum Programmieren Üben, SEUH 2023, 2023.
- [Se20] Seddighi, M.; Allanson, D.; Rothwell, G.; Takrouiri, K.: Study on the use of a combination of IPython Notebook and an industry-standard package in educating a CFD course. Computer Applications in Engineering Education 28/, 2020.
- [Va90] Van Merriënboer, J. J. G.: Strategies for programming instruction in high school: Program completion vs. Program generation. Journal of Educational Computing Research 6/, S. 265–285, 1990.
- [Wi04] Willis, J.: A framework for task-based learning. Longman, Harlow, 2004.
- [WLW15] Wespi, C.; Luthiger, H.; Wilhelm, M.: Mit Aufgabensets Kompetenzaufbau und Kompetenzförderung ermöglichen. Haushalt in Bildung & Forschung 4/4, S. 31–46, 2015.
- [Za19] Zastre, M.: Jupyter Notebook in CS1: An Experience Report. In: Proceedings of the Western Canadian Conference on Computing Education. WCCCE '19, Association for Computing Machinery, Calgary, AB, Canada, 2019.



# Introducing Tablet-Based On-Site E-exams in a Large Software Development Course: An Experience Report

Paula Rachow,<sup>1</sup> Christian Rahe,<sup>2</sup> André van Hoorn<sup>3</sup>

**Abstract:** For introductory programming and software development courses, an electronic examination format with programming tasks in an IDE-like environment seems like the natural choice. However, for our courses at the Universität Hamburg, with up to 450 exam participants, we have so far still conducted paper-based exams that are time-consuming to grade. In the context of modernizing the assessment methods, this paper explores and reflects on introducing tablet-based e-exams in the introductory module “Software Development 2”. This experience report highlights the motivation, challenges, setup, outcomes, and lessons learned of this transition. The implementation of e-exams mirrors industry practices and aligns with the course’s learning objectives. Moreover, the results showcase improved efficiency in grading, reduced logistical challenges, and enhanced accessibility. However, adopting e-exams also introduces challenges, including a dependency on a robust technological infrastructure. This paper offers valuable insights into the advantages and risks for educators seeking to integrate e-exams into their curriculum. Despite initial challenges, the positive outcomes of this transition encourage us to continue integrating e-exams into our curriculum in the future.

**Keywords:** Digital e-exams; Programming education; e-Learning; Automated grading

## 1 Introduction

In the summer semester of 2023, we significantly changed how we assess students in the final exams in the foundational programming module “Software Development 2” at the Universität Hamburg. We shifted from using traditional paper-based exams to on-site tablet-based e-exams with the learning management system Moodle<sup>4</sup> for the summative assessment. We were driven by our desire to make assessments more practical and efficient. Since we were already familiar with using Moodle and had learned from our experience with remote e-exams in 2021 (as a response to the COVID-19 pandemic), we felt prepared for this change. This paper aims to share the reasons behind this shift, explains how we adapted to our specific situation, describes the technology we used, and passes on our learned lessons. We hope to guide other educators who may be considering a similar transition.

---

<sup>1</sup> Universität Hamburg, Vogt-Kölln-Str. 30, 22527 Hamburg, Deutschland, paula.rachow@uni-hamburg.de

<sup>2</sup> Universität Hamburg, Vogt-Kölln-Str. 30, 22527 Hamburg, Deutschland, christian.rahe@uni-hamburg.de

<sup>3</sup> Universität Hamburg, Vogt-Kölln-Str. 30, 22527 Hamburg, Deutschland, andre.van.hoorn@uni-hamburg.de

<sup>4</sup> moodle.org

## 2 Background: Digital assessment

E-exams are digital exams conducted through an e-testing system [Ba21] and they can be held either on-site or remotely. They come in two formats: open-book and closed-book. Open-book e-exams allow students access to reference materials, encouraging them to synthesize information and apply critical thinking. Closed-book e-exams require students to rely solely on their knowledge, skill, and their comprehension of the subject matter.

The whitepaper by Bandelt et al. [Ba21] lists some practical examples of digital exam implementations at different universities, including e-exams. For instance, the University of Zurich shares similarities with our approach, but they rely on a large computer pool, and do not employ tablets as we do [Ha14]. The University of Applied Science Munich provides the automation framework *EXaHM* for digital exams to steer the exam process, like the startup and shutdown of the PCs<sup>5</sup>. Other papers about digital exams often deal with other scenarios, e.g., Bottcher et al. [BT23] report about conducting remote open book exams.

More study was devoted to learning management systems and online programming platforms utilized throughout the semester, such as Artemis [KS18] and OPPSEE [SB23]. However, there is a notable gap in the literature addressing their specific role in exams. Aspects such as automated feedback [Be21] and strategies for more effective manual grading of student exercises [La19] are crucial but are less relevant in the context of e-exams. Gandraß and Schmolitzky [GS19] present another Moodle plugin for programming exercises that looks similar to the CodeRunner plugin used in our setting and described in a later section.

## 3 Our drivers for switching to an e-exam

For us, a strong motivation for implementing e-exams in the context of the course included several compelling expectations.

**Real-world relevance.** E-exams provide a practical assessment environment mirroring real-world software development, where digital tools like compilers and IDEs are commonplace. By providing the students with a compiler for coding tasks, students are assessed in a more realistic environment. It allows them to concentrate on understanding and applying core programming concepts rather than memorizing syntax.

**Constructive alignment.** Unlike traditional paper-based assessments, e-exams closely resemble the learning objectives and the exercise contents. They emphasize comprehension and application, discouraging mere memorization. This tailored approach ensures that the evaluation accurately measures whether students have acquired programming skills.

**Fast feedback.** Grading of e-exams promises a high degree of automation, allowing students to receive timely feedback on their performance, ensuring that students who failed have enough time to prepare for the second exam date in the same semester.

---

<sup>5</sup> [https://www.hm.edu/lehren/kompetenzzentrum\\_digitaales\\_pruefen/EXaHM.de.html](https://www.hm.edu/lehren/kompetenzzentrum_digitaales_pruefen/EXaHM.de.html)

**Reduced effort.** Handling physical exam papers, ensuring fairness in grading, and managing logistical challenges are time-consuming for educators. E-exams can streamline this process, enabling instructors to focus more on teaching and content development.

**Error mitigation.** With e-exams, there are fewer sources for errors while calculating points, and grading the exams as this is done automatically. Furthermore, changes to the accepted solutions or grading scheme can be automatically applied to all submissions.

**Customized question types.** Electronic platforms offer diverse question types beyond multiple-choice, especially coding exercises, allowing for a more comprehensive evaluation of the students' practical programming skills.

**Accessibility and inclusivity.** E-exams support diverse needs with features like adjustable fonts, screen readers, and alternative input and, therefore, promote inclusivity and easy accommodations for students with approved reasonable (special) accommodations.

**Environmentally friendly.** E-exams reduce the need for physical paper, promoting environmental sustainability by reducing waste.

**Data-driven insights.** Electronic platforms often generate analytics and reports, providing educators with valuable data on student performance trends, areas of struggle, and overall class performance.

**Economic.** E-exams eliminate the need for printing — cutting costs on paper, ink, and equipment. There is also less need for physical storage space afterward and no student assistants have to be employed to help with the grading.

## 4 Our setting

### 4.1 Course context

This paper presents a reflection on introducing on-site closed-book e-exams for the “Software Development 2” module at the Universität Hamburg. The module is attended by approximately 450 students each summer semester. The students, typically in their second semester, are enrolled in various bachelor programs, including informatics, human-computer interaction, computing in science, and information systems.

The module provides students with a foundational understanding of software development principles and practices — with a focus on object-orientation. It comprises one weekly lecture and a two-hour in-person exercise session. During these sessions, the students work on programming exercises in pairs or (in the second part of the semester) in groups of four. In addition to these sessions, students complete mandatory weekly Moodle quizzes — including programming tasks — to reinforce their learning.

There are two date options to take the exam each summer semester. The exam has a duration

of 120 minutes. In the summer semester 2023, 203 students participated on the first exam date and 108 students participated on the second exam date.

## 4.2 Lessons learned from a prior remote e-exam

During the COVID-19 pandemic, we had already implemented an e-exam in 2021. However, this e-exam was conducted remotely and in an open-book format. Through this experience, we gained a pool of e-exam questions and identified several critical issues that have informed our decision-making process for the transition to an on-site closed-book e-exam.

**Scalability.** We encountered significant scalability issues with our Moodle platform—particularly related to the CodeRunner plugin and back-end. After all the exams were automatically submitted and the results were calculated, the system experienced a critical breakdown. Luckily, only the student changes from the last few seconds were lost. This incident highlighted the need for a more robust and scalable infrastructure to support the increasing demand for digital assessments. However, it also showed that an e-exam with many students is possible.

**Foundational course.** Our module is a foundational course in the second semester. According to Wollersheim et al. [WMS11], foundational courses primarily target the lower levels of cognitive learning, which encompass ‘knowing’ and ‘understanding’ [Kr75]. Therefore, crafting questions for an open-book exam posed a challenge. It is essential to balance that questions are not easily searchable online and still have a reasonable level of complexity. It convinced us to perform only closed-book tests in the future.

**Cheating.** We encountered multiple instances of cheating. Therefore, we were concerned about the integrity of remote exams, as it is difficult to ascertain who is in front of the computer and whether any unauthorized assistance is present in the room. Especially since the Universität Hamburg does not allow online proctoring of students, which further complicates the issue. Due to these reasons, we preferred to stay with on-site exams.

**Communication.** We realized that direct communication on-site with students is notably easier, especially for delivering announcements and addressing last-minute concerns or queries. It also simplifies communication among exam supervisors, allowing for seamless coordination and immediate response to unforeseen circumstances.

**Responsibility shift.** A take-home exam shifts the responsibility to students for a stable internet connection and suitable testing conditions, potentially disadvantaging some. Despite the tablet being less ergonomically comfortable than the students’ usual setups, on-site e-exams with tablets provide a controlled environment and are a good option when a large pool of computers does not exist.



### 4.3 Procedure and effort comparison

In this section, we compare procedural aspects and effort involved in conducting an e-exam instead of a traditional paper-based exam. Table 1 summarizes the processes/tasks and (estimated) efforts elaborated below. By examining each process step, we aim to provide a detailed insight into the logistical considerations and administrative workload associated with both assessment methods. The shift to e-exams alters time allocation, with more even planning across the semester. During the exam, there are additional technical questions but after the exam, tasks like grading happen faster. Additionally, a significant portion of the additional effort is redirected toward the IT administration.

Tab. 1: Effort of processes and tasks for e-exams compared to paper-based exams.

Task	1 <sup>st</sup> iteration	Future iterations
<b>Preparation</b>	++	+
Creation of exams	++	+
Organization	+	o
<b>Execution</b>	+	+
Supervision	+	+
<b>Post-processing</b>	-	--
Grading	-	--
Analysis	--	--
Exam review	-	--

- - significantly less effort, - less effort, o same effort, + more effort, ++ significantly more effort

#### 4.3.1 Preparation

Creating exam questions is not more complex than with a paper-based exam in L<sup>A</sup>T<sub>E</sub>X. While Moodle allows for more complex code tasks, the assurance of the automatic evaluation induces more effort during setup, in addition to the effort of the course creation, configuration settings, and student enrollment. However, the organization of the printing is no longer necessary. In addition, the tasks can be randomized and random questions can be drawn so that the degree of reuse of the exam tasks is high.

Managing tablets—updating software, charging batteries, and preparing the e-exam configuration—also entails a higher overhead. However, in our case, the responsibility for the tablets lies with the IT administration staff.

#### 4.3.2 Execution

We had additional technical assistance from the IT administration team to ensure a smooth process. They provided technical support and facilitated the delivery and distribution of the tablets. For us, the necessity of picking up and carrying paper exams has been eliminated.

Our university only has 210 tablets (iPads) available for e-exams. Therefore, we used additional PC pools. In the future, we plan to extend the lecture hall reservation and let the students write in two cohorts.

### 4.3.3 Post-processing

**Exam grading.** With the paper-based exam, we had to organize the exam grading process, including finding and preparing rooms, graders, and files. The exam grading used to take up to 4 days with two instructors and multiple student assistants. The grading per exam took approximately 35 minutes including an additional review of each grading. Grades were usually published two weeks after the exam. With the e-exam, the grading is done automatically and the publication on the same day.

**Exam review.** For the exam review, we had to look for the students' exams and hand them out. With Moodle, we can just change the permission for the student to be able to see the results. There is also less chance that the grading has to be changed since the grading is performed automatically. Moreover, if we notice a general error in the rules for the automated grading, we can easily reassess all exams automatically.

## 5 Technical realization of the e-exam



Fig. 1: Students taking e-exam.



Fig. 2: The iPad setup with keyboard.

### 5.1 Moodle setup

We implemented our e-exam in Moodle, a popular open-source learning management system. We chose Moodle because the MIN faculty at the Universität Hamburg operates its own instance and we and the students were already familiar with using it during the semester. Each e-exam has a dedicated Moodle course, implemented as a Moodle *test* activity with password protection. Each student has an individual timer of 120 minutes, after which the test terminates automatically. The timer can be extended for groups or individual students, especially for those with approved special accommodations, granting them the necessary extra time for the exam. The setting “Full screen pop-up with some JavaScript security” ensures an undistracted exam environment by hiding chat and menus.

## 5.2 Hardware setup

Students used 9th generation iPads with integrated keyboards for the exam (see Fig. 2) as they are portable and easy to manage. Managed through the Jamf School fully supervised MDM system<sup>6</sup>, these iPads offer precise control over functionalities, with deliberate restrictions on features like the microphone, camera, and auto-fill options. Bluetooth is deactivated to prevent external interferences and the iPad exclusively connects to the eduroam network. Strict DNS filtering controls access to specific domains, allowing connections only to designated university servers. The exam link is easily accessible through a home screen icon (WebClip), ensuring quick deployment. For future reference, the tablet's device ID is transmitted during the exam. To maintain a secure testing environment, browser navigation is disabled during the exam. For the computers, we used the Safe Exam Browser<sup>7</sup> to limit access exclusively to the e-exam.

## 5.3 Question types

We implemented a Moodle test with 33 to 37 questions per exam, summing up to 120 points. The exam takes 120 minutes so the students can estimate that they should take one minute per point. These exams closely mirrored the format of previous paper-based exams. The questions were mostly independent of each other and covered the following question types:

**Single choice.** We implemented single-choice questions through the multiple-choice question type in Moodle. E.g., we quizzed the students about the definitions of the SOLID principles.

**Drop down.** We implemented various drop-down questions: Cloze (gap fill) questions, selectable short answers, and matching questions. Examples are a cloze for the design-by-contract model and matching Java streams to their corresponding for-loop versions.

**CodeRunner.** The CodeRunner<sup>8</sup> is a plugin for Moodle and was already provided by the university. Unlike traditional formats, CodeRunner requires students to write and execute code. By emphasizing the execution of code over memorizing syntax, CodeRunner aligns with the module's emphasis on practical application and critical thinking in software development. Additionally, assessments in the CodeRunner format are equipped with a compiler, allowing students to concentrate on algorithmic logic and problem-solving strategies without becoming entangled in syntactical details. This format proves invaluable in preparing students for the coding challenges they may encounter in their future careers. In addition, we implemented a graphical UML tool with the CodeRunner plugin to test the students' UML knowledge. Fig. 3 shows a CodeRunner example question where the students had to rewrite the code with a lambda expression.

---

<sup>6</sup> <https://www.jamfschool.com/>

<sup>7</sup> <https://safeexambrowser.org>

<sup>8</sup> [https://moodle.org/plugins/qtype\\_coderunner](https://moodle.org/plugins/qtype_coderunner)

**Lambda-Expressions**

Given is a method call that uses an anonymous inner class to implement the **functional interface** `MessageListener`.

```

1  _chatService.addMessageListener(
2      new MessageListener()
3      {
4          @Override
5          public void messageReceived(Person sender, String message)
6          {
7              System.out.println(sender.getName() + ": " + message);
8          }
9      }
10 );

```

**Task:**

Rewrite the source code to use a **lambda expression** instead.

**Answer:** (penalty regime: 0 %)

Reset answer

```

1  _chatService.addMessageListener((Person sender, String message) ->
2      System.out.println(sender.getName() + ": " + message));

```

Precheck

**Hints:**

- If the **behavior** of the operation (the content of the expression) has been changed, the assignment will score **0 points**.

Fig. 3: A programming question about lambda expressions.

**Excluded question types.** We excluded question types like drag-and-drop because they were not consistently working on the tablet. We also excluded the “Random Short-Answer Matching question type” in the second exam because there was a bug with the evaluation<sup>9</sup>. Moreover, we did not use free text or similar question types because we wanted to be able to evaluate everything automatically.

## 5.4 Moodle extensions

We implemented minor UI enhancements using a Moodle text block in the exam course’s sidebar, containing a *script* tag and incorporating browser-side JavaScript for page modification. One such improvement involved displaying each student’s name and ID number at the top of the browser window. This streamlined the identity verification process for exam supervisors, eliminating the need for students to navigate or scroll to their profile, thereby

<sup>9</sup> <https://tracker.moodle.org/browse/MDL-76676>

expediting the process. Additionally, we modified the submission timer in Moodle to only display seconds during the last ten minutes, reducing visual distractions.

When a quiz is submitted in Moodle, all answers, including CodeRunner questions with potentially resource-intensive testing logic, are automatically processed for grading. Currently, there is no option to disable this feature. Recognizing the potential strain on our infrastructure from a surge of simultaneous submissions, we took proactive measures to prevent exams from being submitted manually in the first place.

**Preventing auto-submissions.** We configured the exam test activity with the timing option "Attempts must be submitted before time expires, or they are not counted". Using JavaScript injection, we prevented the timer from expiring automatically, thus avoiding premature grading. This way, for students with an expiring timer, no automatic grading was initiated. Students were informed in advance that their exam status would show as "Never submitted".

**Preventing manual submissions.** In our exam setup, students were able to exit the exam early. To prevent unintended automatic grading while others were still working, we disabled the "Submit" button. Instead, we instructed students to seek a supervisor's assistance for submission. The supervisor would then apply a user override to conclude the exam, resulting in the "Never submitted" status and averting automatic grading.

## 5.5 CodeRunner extensions

The Java template in CodeRunner combines student answer code and test code in a single compilation unit. If the student code doesn't compile or the test code accesses missing elements, the raw compilation error is presented. This can be confusing for students due to line number offsets and errors referencing code they have neither written nor seen before.

**Missing methods.** In exams, we consider it essential to award part of a question's points for incomplete answers wherever possible. To enable testing of student code with only a subset of required methods, we utilized the Java Reflection API for property accesses, constructor, and method calls. This approach allows specific test cases to fail with a runtime exception, rather than rendering the entire answer invalid.

**Compilation errors.** For the majority of code questions, compilation was required to receive credit. In some cases, however, we wanted to recognize partially correct answers even if they had syntax errors. To ensure we could still run test code if the student code did not compile, we inserted the student submission into our CodeRunner test template as a string and compiled it in memory using the *JavaCompiler* class. This approach allowed us to analyze properties of the submitted source code, e.g. to award partial credit for the presence of certain methods or constructs, even if we could not test its runtime behavior.

Introducing an additional computational cost per question granted us valuable flexibility in question design. For instance, in one question, students needed to identify and fix three

separate compilation errors in a provided code snippet. We were able to award partial credit even if errors persisted in the student's code.

## 6 Lessons learned

In this section, we explore lessons learned from implementing e-exams, covering general and technical insights, each rooted in one of the expectations that led to this transition.

**Real-world relevance:** We were able to pose more complex coding tasks and the students were able to use a compiler to iteratively solve them. An IDE would have been even better and closer to practical programming, but it also comes with additional effort.

**Constructive alignment:** The students were already familiar with the type of questions and the environment from the Moodle quizzes during the semester and could focus on the questions rather than the technical infrastructure.

**Fast feedback:** If there are no complications in grading — such as server overload or solutions — the grades can be published on the same day.

**Reduced effort:** Efficiency improved significantly with reduced grading efforts. However, there is an increased effort for the initial question creation, mainly because we wanted to be able to grade each question automatically. However, we hope that the incorporation of randomization rendered questions highly reusable. For the coding questions, the students' answers usually must be compilable for us to grade them automatically. While this poses a slight disadvantage for students with partially correct answers, we believe it remains feasible and fair, thanks to the compiler's availability and unlimited compiler runs for the students.

There was a considerable demand for IT system administration. To further optimize the process, dedicated exam rooms with proper technological infrastructure could mitigate technical issues and provide a controlled, focused examination environment.

**Error mitigation:** We enhanced accuracy by eliminating error-prone steps, including manual grading, manual data calculation, and manual transfer to Excel and Stine<sup>10</sup>, the information system of the Universität Hamburg. This transition to automated grading significantly enhanced accuracy in the assessment process. This also became obvious when there were no grade corrections after the exam review and no significant changes in the automatic assessment.

**Customized question types:** Moodle questions offer students a more intuitive answering experience by making the desired type of answer clearer. Moodle questions are also easier for instructors to create compared to paper-based exams in L<sup>A</sup>T<sub>E</sub>X, which can be more time-consuming and may not provide the same level of customization.

---

<sup>10</sup> <https://www.stine.uni-hamburg.de/>

**Accessibility and inclusivity:** While we acknowledge the advantages, we did not need it for the initial two e-exam instances.

**Environmentally friendly:** For the previous exams, we had to print 10,500 pages for the two exams that are now no longer necessary.

**Data-driven insights:** Moodle automatically provides the facility index and discriminative efficiency for each question<sup>11</sup>. It also indicates which questions need a review depending on the grading. Currently, the grade distribution appears to be absent and there is only a point distribution diagram in increments of ten points.

**Economic:** There was no need to pay any student assistants to help with the grading.

## 7 Risks and Constraints

Despite the many advantages of e-exams, there are also additional risks and constraints:

**Strong dependency on infrastructure.** Successful e-exams rely heavily on technological components such as tablets, Wi-Fi, platforms like Moodle, and authentication services, all requiring reliable operation. In large organizations like universities, avoiding maintenance during exams poses a challenge. In our case, a pre-scheduled and non-reschedulable maintenance window coincided with the first exam. Fortunately, all systems operated smoothly on the day. Going forward, we have proactively planned around maintenance dates to preempt any potential disruptions to the examination process.

**Limited infrastructure capacity (scalability):** Implementing e-exams places a significant demand on the infrastructure. The system must be robust enough to handle a large number of students simultaneously accessing resources, submitting responses, and interacting with the assessment platform. Scaling up the infrastructure to accommodate a surge in usage during exam periods is essential to prevent potential technical bottlenecks.

**Capacity of devices, rooms, and resources:** With only 210 available tablets, accommodating more students necessitates a sequential exam approach which takes more time and can potentially lead to scheduling conflicts. This year, we had to let some students participate in the PC pool because we could not extend the lecture hall reservation. For the future, we booked double the amount of time to allow for two cohorts.

**Additional error sources:** In e-exams, potential errors can stem from technical settings and human factors. For instance, incorrect configurations or platform settings can disrupt the exam process. Additionally, students may face challenges with passwords, leading to delays. To ensure a smooth assessment, educators must carefully set up the digital exam environment and have plans to address any unforeseen issues swiftly.

<sup>11</sup> [https://docs.moodle.org/403/en/Quiz\\_statistics\\_report](https://docs.moodle.org/403/en/Quiz_statistics_report)

## 8 Conclusion

In conclusion, introducing e-exams in our software development course has proven highly beneficial. Positive student feedback reaffirms the effectiveness of this transition aligning with the findings of Martin et al. [Ma22] that digital formats reduce stress and anxiety. Notably, we observed no significant differences in grade distribution compared to previous paper-based exams. By entrusting the technical tasks and responsibilities to the dedicated team at the university, we have significantly reduced our administrative workload. This shift allows us to concentrate our efforts on the core aspects of teaching and content refinement. Looking ahead, we anticipate that the long-term benefits will outweigh the initial implementation expenses.

## References

- [Ba21] Bandtel, Matthias et al.: Digitale Prüfungen in der Hochschule. Hochschulforum Digitalisierung, 2021.
- [Be21] Bernius, Jan Philip et al.: A machine learning approach for suggesting feedback in textual exercises in large courses. In: Proc. 8th L-AT-S. 2021.
- [BT23] Böttcher, A.; Thurner, V.: Digitale Prüfungen für Softwareentwicklung im “Bring Your Own Device, Open Book, Open Web”-Format. In: SEUH 2023. 2023.
- [GS19] Gandraß, N.; Schmolitzky, A.: Automatisierte Bewertung von Java-Programmieraufgaben im Rahmen einer Moodle E-Learning Plattform. In: Proc. ABP. 2019.
- [Ha14] Halbherr, Tobias et al.: Making examinations more valid, meaningful and motivating: The online exams service at ETH Zurich. In: EUNIS Journal of Higher Education. 2014.
- [Kr75] Krathwohl, David R et al.: Taxonomie von Lernzielen im affektiven Bereich. Beltz, 1975.
- [KS18] Krusche, S.; Seitz, A.: Artemis: An automatic assessment management system for interactive learning. In: Proc. 49th ACM SIGCSE. 2018.
- [La19] Laß, Christopher et al.: Stager: Simplifying the Manual Assessment of Programming Exercises. In: SEUH 2019. 2019.
- [Ma22] Martin, Robert J et al.: Digital, Online, Take-Home–University Students’ Attitude towards Different Examination Formats. In: IEEE GeCon. 2022.
- [SB23] Schmolitzky, A.; Burau, H.: OPPSEE-Eine Online-Plattform zum Programmieren Üben. In: SEUH 2023. 2023.
- [WMS11] Wollersheim, H.-W.; März, M.; Schminder, J.: Digitale Prüfungsformate. Zum Wandel von Prüfungskultur und Prüfungspraxis in modularisierten Studiengängen. Zeitschrift für Pädagogik 57/3, 2011.



## Session 2



# EvalQuiz — LLM-based Automated Generation of Self-Assessment Quizzes in Software Engineering Education

Niklas Meißner <sup>1</sup>, Sandro Speth <sup>1</sup>, Julian Kieslinger<sup>1</sup>, Steffen Becker <sup>1</sup>

**Abstract:** Self-assessment quizzes after lectures, educational videos, or chapters are a commonly used method in software engineering (SE) education to allow students to test the knowledge they have gained. However, creating these quizzes is time-consuming, cognitively exhausting, and complex, as an expert in the field needs to create the quizzes and review the lecture material for validity. Therefore, this paper presents a concept to automatically generate self-assessment quizzes based on lecture material using a large language model (LLM) to reduce lecturers' workload and simplify the general quiz creation process. The developed prototype was handed to experts, who subsequently evaluated the approach. The results show that automatic quiz generation saves time, and the quizzes cover the delivered lecture material well. However, the generated quizzes often lack originality and versatility. Therefore, further prompt engineering might be required to achieve more elaborate results.

**Keywords:** Self-Assessment; Software Engineering Education; Automatic Question Generation; GPT-4; Prompt Engineering

## 1 Introduction, Motivation, and Research Questions

In today's higher education, self-assessment quizzes are increasingly used to provide students with tests of what they previously should have learned [FB89]. Also, in the area of software engineering (SE) education at universities, these teaching methods are used and integrated into lectures. Self-assessment quizzes are usually separate, independent questions, such as multiple-choice questions (MCQs), and only offer students a test without any external assessment or feedback [Bo13]. Quizzes are also a growing component of digital education that is increasingly being delivered with course content in learning management systems (LMSs). Many LMSs allow manual insertion of self-assessments in the form of quizzes. This allows instructors to distribute their created self-assessment quizzes in lectures and remotely on the LMSs [Sw17]. However, creating self-assessments is challenging and can be tedious, especially because MCQs are time-consuming to create [Ga19]. It is not easy to come up with quizzes that are relevant, interesting, and of the right difficulty level, as it requires creativity and results in cognitive drain. Lecturers often lack time to create and offer self-assessments. Also, in higher education, too little emphasis is placed on student self-assessment, although it benefits students [An19]. In addition, self-assessment is not considered essential as it is less conventional than traditional forms of teaching. The integration and flexibility of the quizzes in the LMSs are relatively poor.

---

<sup>1</sup> University of Stuttgart, Institute of Software Engineering, Germany {niklas.meissner,sandro.speth,julian.kieslinger,steffen.becker}@iste.uni-stuttgart.de

Previous research has shown the potential of LLMs such as *GPT-3.5-turbo* for creating programming exercises in higher education [SMB23]. Based on the positive results of the programming exercise creation using *GPT-3.5-turbo*, the GPT LLM was selected in this paper, and the generation pipeline for creating self-assessment quizzes such as MCQs was developed. Thus, *EvalQuiz* conceptually serves as a pipeline for the automated generation of self-assessment quizzes using OpenAI’s *GPT-4* [Op23] API. Therefore, our research questions are the following:

**RQ1:** “How can the process of creating self-assessment quizzes in SE courses be automated to support lecturers in their teaching?”

**RQ2:** “How suitable are GPT-generated self-assessment quizzes in the field of SE in higher education?”

In this paper, we tackle these problems and introduce *EvalQuiz*, a concept for the automated generation of self-assessment quizzes based on lecture material using large language models (LLMs). By answering these research questions, we aim to provide the basis for further research into the automated generation of self-assessment quizzes using LLMs. A demo video of the implemented *EvalQuiz* prototype is available on YouTube<sup>2</sup>.

The remainder is structured as follows: In Section 2, we present the concept of quiz generation with the corresponding pipeline, topic modeling, and keyword extraction, prompt engineering and question generation, assembling and validation of generated quizzes, and evaluating their quality. The prototypical implementation of the concept is described in Section 3. Afterward, in Section 4, we examine our evaluation, including the process, results, and discussion. We discuss our threats to validity in Section 5 and outline related work in Section 6. Finally, we conclude the paper in Section 7.

## 2 Quiz Generation Concept

To ensure that the generation of questions is coherent from a didactic point of view regarding structure, content, and desired capabilities, close cooperation with experts in higher education didactics at the University of Stuttgart was established as part of the research. This involved focusing on the problems described above and deriving methods for the automated generation of MCQs on this basis. This chapter contains the concept of *EvalQuiz*, including the generation pipeline and the individual steps from uploading the lecture material to the final generated MCQs.

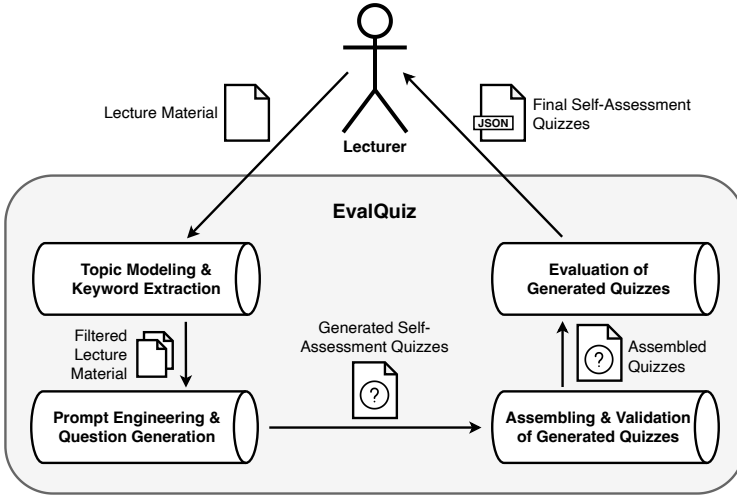


Fig. 1: Pipeline of quiz generation process.

## 2.1 Generation Pipeline

Figure 1 shows the conceptual structure of the self-assessment quiz generation pipeline. The lecturer provides lecture material and adds it to the pipeline. Lecturers can also specify configurations regarding how the questions should be generated or what focus they should have based on keywords and their relations and capabilities according to Bloom’s taxonomy [BK20]. The first processing step is topic modeling & keyword extraction (Section 2.2), in which keywords for the quizzes are filtered, and texts from the lecture material are summarized. The filtered lecture material is then transferred to the prompt engineering stage (Section 2.3), where a prompt for the LLM is created, and the question generation is started. The generated quizzes are inserted into the assembling & validation stage of the pipeline (Section 2.4), where the generated questions are validated. Subsequently, the quality of the generated questions can be evaluated by lecturers and regenerated in case of poor generation quality (Section 2.5). To ensure a uniform listing and output of the self-assessment quizzes, we use the *JSON* format for the generated quizzes.

## 2.2 Topic Modeling and Keyword Extraction

LLMs only support prompts up to a particular token size. Unfortunately, lecture materials can exceed these boundaries, so reducing the input as much as possible without losing relevant information is crucial. This stage filters input documents to select parts of specific topics. There are various methods for processing large quantities of data as input. We assume

<sup>2</sup> <https://www.youtube.com/watch?v=kC-9e2Bh4nQ>

Type	Amount	Example
System message	1	“You are a question-generation assistant that supports generating questions in multiple fixed formats.”
Few-shot examples	n	Configurations & filtered input; “Your goal is to use the given markdown formatted text input to generate a question in the following JSON format: <i>example</i> ”
Query message	1	“Give your answer in the specified JSON format.”

Tab. 1: Composition of question generation prompts.

a lecture material is topic-wise and conceptionally divided into chapters. One possibility is to split up the chapters and process each part individually. However, cross-references between different chapters become complex. It is essential to maintain the coherence of content with the keywords between chapters, so an algorithm is needed that produces a subset of the input text containing information of the related topics in a reduced form. First, the lecture material is uniformly transformed into *markdown* to simplify the search for keywords. Then, the *Word2Vec* [Ch17] topic model filters and summarizes by keywords to consider the coherence of sentences, which is a strength of this topic model. *Word2Vec* uses a neural network model to learn word associations, works on a word-vector representation, and encodes words in a multidimensional vector space. This allows for synthesizing information based on sentences containing keywords summarized for a specific context.

### 2.3 Prompt Engineering & Question Generation

The filtered lecture material from the first stage and the specified configuration of the lecturers serve as input for the prompt engineering and question generation stage. This step composes a prompt for the LLM. In our prototype, we use OpenAI’s *GPT-4* [Op23] API to generate the MCQs. The prompts are composed of a *system message*, which provides the frame of the prompt, then *few-shot examples*, which includes the content of the request and contains sample answers and formats, and a *query message*, which gives a concrete instruction for action. The structure of the prompts with examples of the individual components is shown in Table 1. The *system message* contains general instructions for the LLM to show what is important in the following messages and what needs to be done. The *few-shot examples* consist of the lecturer configuration and the filtered lecture material. It also contains the desired competence level [BK20], based on which the quiz will be generated. As the questions ultimately have to be formatted uniformly in *JSON* format, the few-shot examples also contain a *JSON* example that the LLM should be guided by. We set placeholders for the question-and-answer texts and the distractors in the provided *JSON* example for the creation of MCQs, which the LLM must replace with the generated question-and-answer options. The *query message* then emphasizes the generation based on the *JSON* example. Finally, the composed prompt is transmitted isolated to the LLM, i.e., in our prototype, the *GPT-4* API, and thus the generation of questions starts.

```

1 {
2   "question_type": "multiple_choice",
3   "generation_result": {
4     "question_text": "What does SDLC stands for?",
5     "answer_text": "Software Development Life Cycle",
6     "distractor_text": ["System Design Life Cycle", "Software Design Life
7       Cycle", "System Development Life Cycle"]
8   }
9 }

```

List. 1: Multiple-choice example as defined by *EvalQuiz* specification (JSON).

## 2.4 Assembling & Validation of Generated Quizzes

Once the LLM has generated the set of quizzes, they are tested and validated for specification matching in this stage. To use the output directly and deliver it back to the lecturer, the output must match the required specification, i.e., the *JSON* format with the corresponding attributes. The required specification of the output format and attributes used in *EvalQuiz* is shown as an example in Listing 1. The format of the generated questions is checked by an algorithm that matches the *JSON* output with the given specification. For this purpose, the generated LLM output is divided into three categories. (1) The output format matches the specification and can be used directly; (2) the output format roughly matches the specification, but further modifications need to be made, e.g., the attributes are misnamed. An improvement can be achieved, i.e., by re-executing the request so that the LLM generates a new response to the same prompt, as already experienced in previous work [SMB23]. (3) The output format does not match the specification, and the prompt must be rebuilt and revised in the second stage to improve the output.

## 2.5 Evaluation of Generated Quizzes

The last stage evaluates the quality of the generated and validated quizzes. Here, another prompt engineering process is used to transmit the generated questions back to the LLM with a new prompt and requirements for their evaluation. For instance, the question evaluation could be configured to focus on selected question-wording guidelines and check the generated questions against these. (1) The question uses simple language and is easy to understand. (2) The question should have a simple structure and avoid double negatives. (3) Suggestive questions should be avoided as they may be too obvious. (4) The questions have a temporal reference, and dates and time periods are stated precisely. (5) Use of concise answer categories. Closed questions should have disjunctive (non-overlapping) answer categories. (6) Unclear and necessary terms should be explained. (7) Terms associated with strong opinions and emotions should be avoided. (8) Questions must be unambiguous, leaving no room for individual interpretation. The questions are evaluated on these categories, and

based on this, a decision is made on whether a question can be sent to the lecturer or whether it needs to be regenerated. All questions that have been fully evaluated and accepted are then passed on to the lecturer. In the *EvalQuiz* prototype (Section 3), the automated evaluation by the LLM of the generated questions described here is only partially implemented, requiring further specification of the respective criteria (1-8).

### 3 EvalQuiz Prototype

The prototype developed encompasses the concept described in Section 2 and is used to test and evaluate the concept described in Section 4. The implementation included three areas:

1. *Material server*<sup>3</sup>: handles the storage and management of the lecture material.
2. *Pipeline server*<sup>4</sup>: includes the described concept pipeline. This uses the stored lecture material and generates the respective MCQs.
3. *Frontend*<sup>5</sup>: provides faculty members a user interface (UI) to interact with the system, upload lecture material, and receive the generated questions.

The *frontend* is implemented using *React*, while the *backend* was developed using *Python*. The source code is open-source and available on GitHub and documented<sup>6</sup>. Based on experience in previous work [SMB23], we decided to use OpenAI's *GPT-4* as LLM for this prototype, which we call via its API. For space constraints, this paper will not further discuss our implementation methods. Furthermore, the prototype has not yet fully implemented all the functions described in the pipeline concept, e.g., the question evaluation stage is only partially implemented. However, a demo video of the prototype is available on YouTube<sup>2</sup>.

### 4 Evaluation, Results and Discussion

This chapter outlines the evaluation of the concept. For this purpose, the implemented prototype was used and distributed to faculty members. The participants were obtained based on our professional network, all of them working in the field of SE education. The following subsections describe the evaluation process (Section 4.1), provide demographic information about the participants (Section 4.2), report the results (Section 4.3), and finally discuss the results (Section 4.4).



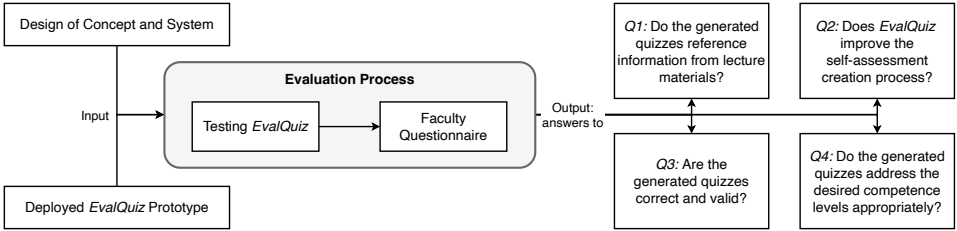


Fig. 2: Evaluation process.

#### 4.1 Evaluation Process

The evaluation process is illustrated in Figure 2. The participants received an instruction, which included a description of the concept (available online<sup>7</sup>) and documentation of the prototype. In addition, we publicly deployed a version of *EvalQuiz* so that the participants could directly focus exclusively on testing the functionalities. They were invited to upload lecture material and generate questions in the deployed version of *EvalQuiz* and were then asked to describe their interaction, experiences, and impressions in the provided questionnaire. The questions and results are available online on Zenodo<sup>8</sup> as well as on the Microsoft Forms summary page<sup>9</sup>. The results of the survey are then used to answer the questions of whether (*Q1*) the generated quizzes reflect the uploaded lecture material, (*Q2*) *EvalQuiz* improves the self-assessment process, and (*Q3*) the generated quizzes are correct and valid.

#### 4.2 Participants

In September and October 2023, six participants took part in evaluating and completing the questionnaire. The demographic information is included in the questionnaire responses<sup>8</sup>. Three participants (50%) reported being in the role of professor or lecturer, while two participants (33%) represented PhD students and one (17%) PostDoc position. Of the participants, two (33%) are in the research and teaching area of software engineering, two (33%) in computer science, one (17%) in machine learning, and one (17%) in cloud computing and systems architecture. Three participants (50%) indicated, that they are “*very experienced*” in creating assessments, two (33%) answered they are “*experienced*”, and one (17%) chose “*moderate*”.

<sup>3</sup> <https://github.com/MEITREX/evalquiz-material-server>

<sup>4</sup> <https://github.com/MEITREX/evalquiz-pipeline-server>

<sup>5</sup> <https://github.com/MEITREX/evalquiz-client-react>

<sup>6</sup> <https://meitrex.github.io/evalquiz-material-server/>,  
<https://meitrex.github.io/evalquiz-pipeline-server>

<sup>7</sup> <https://demo.hedgedoc.org/s/ZuHxrgpSb>

<sup>8</sup> <https://zenodo.org/doi/10.5281/zenodo.10040085>

<sup>9</sup> <https://bit.ly/evalquiz-questionnaire>

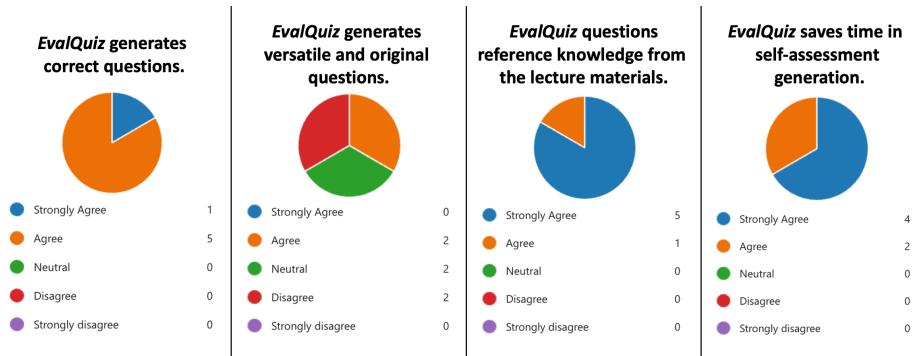


Fig. 3: Questionnaire results.

### 4.3 Results

Regarding general questions about self-assessment in SE education, the participants indicated that MCQs are most commonly used in their lectures (83%), followed by group discussion exercise slides (“blue slides”) (67%), where students can test their knowledge during the lecture, and mock exams (67%). All stated that time constraints are the biggest challenges in self-assessment creation, followed by creativity (33%). 67% of them claimed that there would not be enough self-assessment in their lectures.

All participants responded that they had uploaded lecture slides, although one had also uploaded a paper. Regarding using *EvalQuiz*, all participants indicated that the tool generates correct questions. In addition, they agreed that *EvalQuiz* generates questions based on the uploaded lecture material’s content and that using the tool saves time in self-assessment generation. However, participants were divided about whether *EvalQuiz* creates versatile and original questions. There, opinions were split to “agree” (33%), “neutral” (33%), and “disagree” (33%). Figure 3 depicts these results. Nevertheless, half of the participants agreed that the generation of questions is intuitive, and one disagreed (two neutrals). Moreover, half of the participants agreed that *EvalQuiz* gives the possibility to steer the question generation in the desired direction. The other half had a neutral opinion.

In response to the question of where *EvalQuiz* still has difficulties in the generation, the participants answered that some MCQs have incomprehensible or erroneous distractors. There were some suggestions for improvement, like the support of PDF files, which is currently still limited. In addition, further improvements in the generation of distractors and improvements of questions in the application context were requested. Further, future features were proposed, such as more question types, more variety and diversity in the questions, as well as a user intervention in the prompt engineering of the tool. Finally, participants mentioned that the *EvalQuiz* implementation is a good initial minimum viable product and that they would like to use it for their lectures. The detailed questionnaire results are documented on Zenodo<sup>8</sup> and Microsoft Forms<sup>9</sup>.

## 4.4 Discussion

The evaluation shows that *EvalQuiz* improves the biggest bottleneck in self-assessment creation: *time*. Feedback on the quality of the quiz generation is positive, while there are drawbacks in control and intuitiveness. The participants liked the iterative generation approach but wished they had more control. However, different concepts of each pipeline step influence the generation, which makes it more complex to give more control to the users. Overall, the uploaded lecture materials strongly influence the generation process.

Furthermore, the results from Section 4.3 can be used to answer the questions from the evaluation process, as (*Q1*) the generated questions using *EvalQuiz* address information from the lecture material, (*Q2*) *EvalQuiz* improves the self-assessment creation process, especially in the time dimension, and (*Q3*) the generated quizzes are correct and valid. The evaluation results provide valuable feedback, even if large-scale evaluations are required for deeper insights. Individual comments recommend possible further improvements. One participant states that he “would actually use [the system] to get some questions”. The evaluation indicates that *EvalQuiz* has the potential to be used in education and can support lecturers in creating self-assessment quizzes. However, we expected our participants to use lecture materials besides lecture slides. Thus, books, wikis, exercises, and other lecture materials remain unexplored.

## 5 Threats to Validity

In this chapter, we discuss potential threats to the validity of our study. First, we discuss the internal validity, then external validity, and finally, the construct validity.

*Internal Validity:* The evaluation participants could know each other since the tool was only distributed in the authors’ professional network. Therefore, participants could have talked to each other before testing *EvalQuiz* and filling out the questionnaire. However, we do not assume that the participants talked to each other beforehand since all of them used different lecture materials for the evaluation, and all of them were approached around the same time, so a short-term exchange is considered unlikely. In addition, the participants have different roles and backgrounds at the universities, so there could be differences in terms and wording. Nonetheless, we attempted to define all the essential terms in the questionnaire and describe them with examples.

*External Validity:* Performing a powerful evaluation of the concept and the tool with six participants is insufficient to make a representative statement about its general suitability. Nevertheless, we tried asking participants with different roles and backgrounds to get a broad spectrum of feedback and educational content. In general, participant responses may not be complete for our research questions, and there may be other opinions and application areas where *EvalQuiz* performs differently. Nevertheless, our results are valid answers to the two research questions but might require further investigation.

*Construct Validity:* We prepared the questionnaire questions before distributing them to the participants and ran a test run, which we excluded from the results. While conducting the evaluation of the tool and completing the questionnaire, we did not change any questions or modify any functionalities or attributes of the tool. Therefore, we assume that there is no threat to construct validity.

## 6 Related Work

Previous works focus on specific parts of the pipeline for the automated generation of self-assessment quizzes described in this paper. This chapter focuses on the related work of the individual pipeline elements and distinguishes them from this work.

Previous work by Majumder and Saha [MS15] focused on sentence selection from the input text, among other things, similar to how it is done in this work. They select sentences for the MCQ based on keywords, grammatical structures, the position and length of sentences, as well as pronouns, the completeness of context, and the similarities of definitions. While the approach is similar to our chosen one, it differs in the use of the tools, as in our approach, the input format is first transformed to markdown source code and then filtered for keywords exclusively with the use of *GPT-4*. In addition to exploring input selection, Majumder and Saha designed an MCQ generation system with topic modeling based on an existing topic modeling tool. For the generation, a reference set of MCQs is first collected from a domain. The reference questions are converted from an interrogative to an assertive form. The parse tree structure of the reference sets is analyzed and matched with the input text to find more candidates for MCQs. However, the authors did not use LLM to create the MCQ, and the approach has a few limitations, e.g., sentences could be selected that are too general to answer or omit information that is required for an explicit answer. Also, the generated distractors could be considered valid answers to the question, making the questions flawed.

Araki et al. [Ar16] present two strategies for generating question sentences from multiple input sentences using semantic text analysis. Their method works with texts annotated by experts. Distractor generation is implemented by searching for annotations similar to the answer. A limitation is that questions and distractors are sometimes prone to grammatical errors [Ar16]. While using a different approach for the generation, the authors also tested their approach only in the field of biology. Generalizability to other domains, such as SE, is unclear. Klein and Nabi [KN19] also developed a system for generating questions and answers using transformer models and argued that the two tasks are closely related and can benefit from each other. The authors use BERT and GPT-2, respectively, and argue that GPT-2 is suitable for generating questions and BERT is suitable for answering questions [KN19]. However, the authors did not investigate any generations of questions based on the lecture material utilized, and furthermore, the chosen LLMs are outdated nowadays.

Tsai et al. [TCY21] present a method for retrieving sentences with relevant keywords in a Python learning course using BERT and creating self-assessment questions using GPT-2. They stated that BERT is fine-tuned for better domain-specific keyword recognition, which improves the performance of correct keyword recognition from 94% to 98% [TCY21]. However, examples of falsely recognized keywords in a sentence are not transparently highlighted. Also, the domain-specific fine-tuning does not apply to lecture materials outside the domain of Python programming. Also, Nguyen et al. [Ng22] present and evaluate an approach for creating MCQs in a data science course using MOOCubeX as a topic modeling tool [Ng22]. However, another LLM (Google T5) is used, and the authors do not provide a fixed format for the generated questions.

## 7 Conclusion & Future Work

In this paper, we present a concept for the automated generation of self-assessment quizzes in the software engineering (SE) education domain. For this purpose, we built a pipeline that collects lecture material from lecturers, processes them using topic modeling and keyword extraction, generates a prompt, and thus generates, validates, and evaluates questions about the material. The final generated self-assessment quizzes are then delivered to the lecturer as output. The results of the evaluation conducted on the concept and implemented prototype show that lecturers perceived the generation of questions as time-saving and were able to generate questions quickly. Furthermore, most lecturers stated that the generation of questions is reliable, and all of them stated that *EvalQuiz* generates correct questions. Based on these results, the first research question (*RQ1*) can be answered in a positive way, and *EvalQuiz* supports lecturers in creating self-assessment quizzes. The second research question (*RQ2*) can be answered by indicating that all lecturers stated that the generated questions fit the presented lecture material. Nevertheless, lecturers were undecided whether the generated questions were original and versatile, which could make it difficult to use the generation for many similar SE materials. However, based on the results, *RQ2* can also be considered with a positive result and the generation of self-assessment quizzes using *GPT-4* is suitable in the field of SE in higher education.

In future work, we investigate how students perceive the automated self-assessment quizzes and evaluate them accordingly. We also aim to optimize the generation of self-assessment quizzes to generate more original and versatile quizzes. We also integrate the pipeline into an intelligent tutoring system concept [MSB23] to gain sufficient student analysis items through the automated generation of MCQs without requiring any extra effort from faculty members. Furthermore, we test the performance of other open-source LLMs.

### Acknowledgement

We acknowledge the support of the Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg (MWK, Ministry of Science, Research and the Arts Baden-Württemberg under Az. 33-7533-9-19/54/5) in Künstliche Intelligenz & Gesellschaft: Reflecting Intelligent Systems for Diversity, Demography and Democracy (IRIS3D) and the support by the Interchange Forum for Reflecting on Intelligent Systems (IRIS) at the University of Stuttgart.

## References

- [An19] Andrade, H. L.: A Critical Review of Research on Student Self-Assessment. *Frontiers in Education* 4/, 2019, ISSN: 2504-284X.
- [Ar16] Araki, J. et al.: Generating Questions and Multiple-Choice Answers using Semantic Analysis of Texts. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, Osaka, Japan, pp. 1125–1136, 2016.
- [BK20] Bloom, B. S.; Krathwohl, D. R.: *Taxonomy of educational objectives: The classification of educational goals. Book 1, Cognitive domain*. longman, 2020.
- [Bo13] Boud, D.: *Enhancing learning through self-assessment*. Routledge, 2013.
- [Ch17] Church, K. W.: *Word2Vec*. *Natural Language Engineering* 23/1, 2017.
- [FB89] Falchikov, N.; Boud, D.: Student Self-Assessment in Higher Education: A Meta-Analysis. *Review of Educational Research* 59/4, pp. 395–430, 1989.
- [Ga19] Gamage, S. H. P. W. et al.: Optimising Moodle quizzes for online assessments. *International Journal of STEM Education* 6/1, p. 27, Aug. 2019.
- [KN19] Klein, T.; Nabi, M.: Learning to answer by learning to ask: Getting the best of gpt-2 and bert worlds. *arXiv preprint arXiv:1911.02365*, 2019.
- [MS15] Majumder, M.; Saha, S. K.: A System for Generating Multiple Choice Questions: With a Novel Approach for Sentence Selection. In: *Proceedings of the 2nd NLP-TEA workshop*. Pp. 64–72, 2015.
- [MSB23] Meißner, N.; Speth, S.; Breitenbücher, U.: An Intelligent Tutoring System Concept for a Gamified e-Learning Platform for Higher Computer Science Education. In: *Proceedings of SEUH 2023*. GI, Feb. 2023.
- [Ng22] Nguyen, H. A. et al.: Towards Generalized Methods for Automatic Question Generation in Educational Domains. In: *Educating for a New Future: Making Sense of Technology-Enhanced Learning Adoption*. Springer International Publishing, Cham, pp. 272–284, 2022, ISBN: 978-3-031-16290-9.
- [Op23] OpenAI: GPT-4 Technical Report, 2023, arXiv: 2303.08774 [cs.CL].
- [SMB23] Speth, S.; Meißner, N.; Becker, S.: Investigating the Use of AI-Generated Exercises for Beginner and Intermediate Programming Courses: A ChatGPT Case Study. In: *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T)*. Pp. 142–146, 2023.
- [Sw17] Swart, A. J.: Using reflective self-assessments in a learning management system to promote student engagement and academic success. In: *2017 IEEE Global Engineering Education Conference (EDUCON)*. Pp. 175–180, 2017.
- [TCY21] Tsai, D. C. L.; Chang, W. J. W.; Yang, S. J. H.: Short Answer Questions Generation by Fine-Tuning BERT and GPT-2. In: *Proceedings of the 29th International Conference on Computers in Education Conference, ICCE*. 2021.

# TILE and MASS, a retrospective

Steffen Dick<sup>1</sup>, Teresa Dreyer<sup>2</sup>, Christoph Bockisch<sup>3</sup>

**Abstract:** In conjunction with the QPED project, we have developed two teaching tools, MASS, an automated feedback tool for code, and TILE, a test-driven exercise paradigm. Over the course of three different iterations of the same university courses, we have collected data to see what effect MASS and TILE have had on the students who were confronted with both. For this we used a survey in a later module and a diagnostic assessment within the final exam in the module where TILE and MASS are introduced. We found a substantial and statistically significant positive effect in our exam data.

**Keywords:** survey, diagnostics, analysis, teaching testing, teaching software quality

## 1 Introduction

In an attempt to improve our teaching, in particular in early programming-related courses, at Philipps University Marburg and within the QPED-project we have developed tools to help early learners achieve a better understanding of core concepts of object-oriented programming. Furthermore, those very same tools are supposed to sharpen their knowledge of software quality, especially principles of testing programming code with unit-tests. For those purposes, we've developed two major tools: An auto-assessment tool called MASS and an approach for creating tasks for students called TILE.

The *Marburg university auto ASsess System* (MASS)<sup>4</sup> was developed to give students early feedback on their written JAVA-code. Internally, MASS uses established software (like PMD) to generate a proto-feedback. This proto-feedback can then be used to generate a better understandable feedback for early learners. For example, MASS uses the JAVA compiler to check for any syntax-errors and uses the, sometimes hard to understand, output to generate a much easier to understand feedback.

The other tool, the *Test Informed Learning with Examples* approach [VDM22], or TILE for short, suggests that testing can be taught without detracting any valuable time from other important principles that need to be taught to early learners. TILE uses different domains of testing-tasks to accomplish this. For example, instead of using random texts for tasks, messages that insinuate the importance of testing can be used.

---

<sup>1</sup> Philipps Universität Marburg, Germany dickst@informatik.uni-marburg.de

<sup>2</sup> Philipps Universität Marburg, Germany tdreyer@informatik.uni-marburg.de

<sup>3</sup> Philipps Universität Marburg, Germany bockisch@informatik.uni-marburg.de

<sup>4</sup> For more information on MASS visit <https://qppeu.github.io/mass/>

To evaluate these teaching tools, we devised a survey and a diagnostic assessment. The survey was completed by multiple cohorts over the course of three years to get the perspective of the learners themselves. The diagnostic assessment was added to the final exam of the first-year object-oriented programming course, for an objective perspective.

Lastly, we performed several statistical analyses in order to examine the effect of the implementation of TILE and MASS. We analyzed the effect on the survey data as well as on the exam data. We found a substantial effect on the diagnostic assessment as well as limited effects on the scores in the self-assessment data. Therefore, we conclude that TILE and MASS added value to our lecture and were beneficial to our students.

## 2 Data

This section consists of four main parts. As we will be doing four simple linear regressions, the first part of this section will be about the prerequisites to make a linear regression possible. In the second part we will be discussing the results of a survey we gave to students participating in the Programmierpraktikum at Philipps University Marburg which usually takes place in the second semester of their plan of studies. The third part consists of a diagnostic assessment we added to the exam of the first-semester lecture Object oriented Programming (OoP) that consists of an assessment of the students ability to spot quality flaws within a code snippet and writing unit-tests to check this very same code snippet. Both, the survey and the assessment, were continued over the course of three years and the first iteration of these was discussed in [DSB22]. Finally, the fourth part we will be comparing the subjective results of the study to the objective results of the assessment.

### 2.1 Part I: Prerequisites

For a simple linear regression, four assumptions [EGS13, chapter 18.13] should be considered. These assumptions can be formally tested and/or assessed by visually inspecting a relevant plot.

The first assumption, **Linear Effect**, specifies that the data should contain a linear effect. Additional non-linear effects can also be present. In practice, this assumption often is directly derived from the hypothesis and not formally tested.

The property **Independence of Regression Residuals** or more concretely the assumption **(No) Autocorrelation** specifies that after taking the model into account, no association is left between the regression residuals. It is often violated if the model is incomplete, i.e. not all factors that influence the outcome have been taken into account or if there are other unexplained dependencies between the measurement points.

**Homoscedasticity** means that the variance of the regression residuals is not different between different levels of the independent variable, i.e. between the different years.



The fourth and last assumption is the **Normal Distribution of Regression Residuals**. It means that the values differ between observed value and estimated value according to the regression follow a normal distribution. If this assumption is violated, it can sometimes be restored by linearly transforming the data. It is also possible to perform the Linear Regression using bootstrapping instead of the classical variant. Bootstrapped Linear Regression is robust against the normal distribution violation but has slightly varying results between instances.

assumption	Relevant Test	Criterion	Relevant Plot	Criterion
Linear Effect	None	None	Scatter-plot	Close to linear shape of point cloud
Autocorrelation	Durbin-Watson	not significant and statistic DW close to 2 (can range from 0 to 4)	Fitted-vs.-residuals plot	No trend visible in regression residuals
Homoscedasticity	Breusch-Pagan	not significant	Fitted-vs.-residuals plot	At each fitted value, the variance of the residuals looks similar to their shape at other fitted values.
Normal Distribution	Kolmogorov-Smirnov	not significant	Q-Q plot	Most residual values are on straight line, particularly in the middle of the theoretical distributions between -2 and +2 SD on the x-axis

Tab. 1: Overview over chosen assumption tests and relevant plots

Table 1 shows an overview over the assumptions and the assessment methods we have chosen. The choice and interpretation of the relevant plots for the visual inspection and of the formal test for autocorrelation is based on [FMF12, chapter 7.9] and [Co23, chapter 3.1]. For the choice of the test for the homoscedasticity assumption as well as for its assessment, we followed [FW11, chapter 6.5]. We will formally test the normal distribution assumption with the Kolmogorov-Smirnov test based on [EGS13, chapter 10.6.1].

When performing inferential statistics it must be taken into account that the sample size is an important factor for the size of the p-value [EGS13, chapter 8.2 and chapter 10.6]. Because of this, very small effects can become statistically significant if the number of participants is large. This phenomenon is relevant for the interpretation of the results of the linear regression and for the interpretation of the formal tests regarding the assumptions: The tests for formal assumptions test whether there is a significant deviation from the relevant assumption. Therefore, with an increasing sample size, even very small deviations can become statistically significant. Conversely, if the effects in the linear regression are small, very large sample sizes are needed in order for it to become statistically significant. This phenomenon needs to be kept in mind when interpreting the results [EGS13, chapter 8.2 and chapter 10.6]; the p-value should only be considered one part of the picture.

For the statistical evaluation we used the software R (version 4.3.1) using the libraries readxl (version 1.4.3), car (version 3.1-2), lmtest (version 0.9-40), ggplot2 (version 3.4.4)

and ggplotr (version 0.0.6). For aggregation, data rearrangement and manual data cleaning we used Microsoft Excel (Microsoft Office Professional Plus 2019).

2.2 Part II: Self-Assessment

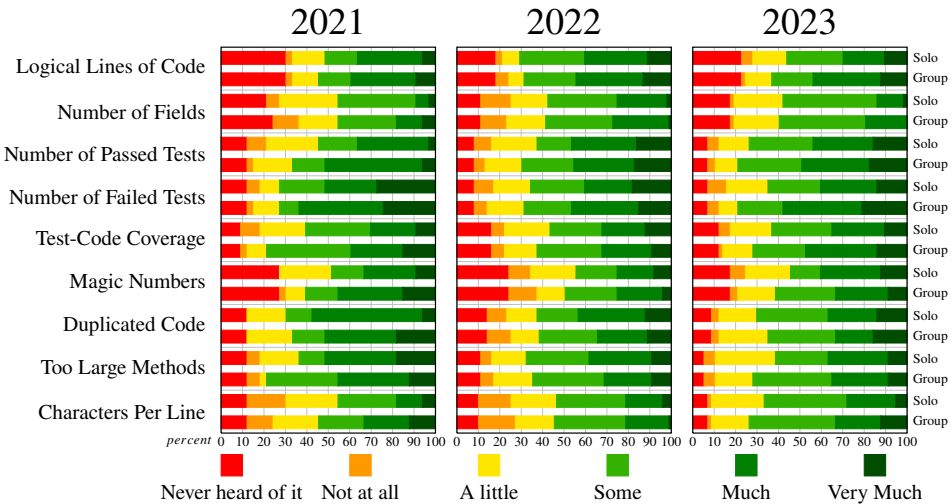


Fig. 1: Participants rating how much attention concepts receive when coding in 2021, 2022 and 2023.

Our first data collection action is based on a survey in which students self-assess how relevant that writing good quality software is for them. We asked the students to rate how much attention they pay to a selection of 9 different software metrics when producing software code in two different contexts: working alone and working within a group. The rating was done on a Likert-scale between one through six, with 1 being the most negative and 6 being the most positive, or from "never heard of it" to "very much" in words. The element "never heard of it" was interpreted as the lowest element on the scale. As described in [DSB22], a selection of software metrics had been made based on easiness of understanding and relevancy to the curriculum of the university of Marburg. This survey was done in the Programmierpraktikum which is a module that takes place in the semester after OoP, so participating students would have been introduced to TILE and MASS. The result of this part of the survey is depicted in Figure 1 sorted by year.

Figure 1 shows the 9 selected software metrics on the left hand side. Every software metric has two lines per year, the upper one for the solo context and the lower one for the group context. Their answers are portrayed within each line as a color coded bar. The negative answers, "never heard of it", "not at all" and "a little", are portrayed in red, orange and yellow respectively. The more positive answers, "some", "much" and "very much" are depicted in darkening shades of green, where "some" has the lightest shade and "very much" the darkest. The percentages are portrayed at the bottom of each individual chart.

As a preliminary examination, we can identify a positive trend: In every succeeding year, the number of students who selected "not at all" for some of the software metrics decreased with Magic Numbers, Logical Lines of Code and Duplicated Code being the exception. Some of the metrics have a little spike in 2022 but are still lower than they were at the start. The number of students who selected "never heard of it" also decreased overall.

Generally, as we can see in Figure 1, the assumption that students pay more attention to software quality metrics in a group context seems to be true. However, Duplicated Code does not follow this trend. In 2021, it can be observed that "a little" has been answered more often in the group context. This continues in 2022 with the "not at all" and "a little" being chosen more often. Even in 2023, "not at all" stays the same but "a little" was answered more often in the group context. At this point in time and with only this much data gathered, an explanation for such a behaviour can only be speculative without further research.

Since we introduced the test-focused TILE concept into the first-semester lecture OoP in the Winter Semester (WS) 21/22, we focus on the software metrics associated with testing. Furthermore, we introduced MASS into the OoP course of WS 22/23 which is also able to provide more individualized feedback on tests written by the students. With both of these tools integrated into the course, we expect to see a higher assessed importance of all software metrics, especially with the testing ones.

	2021		2022		2023	
	<i>Solo</i>	<i>Group</i>	<i>Solo</i>	<i>Group</i>	<i>Solo</i>	<i>Group</i>
<i>Overall</i>	57	64	60	62	63	69
<i>Overall without testing</i>	47	60	59	60	61	66
<i>Only Testing</i>	63	73	62	67	67	77

Tab. 2: Aggregation of green bars by context and year for the three testing metrics

For our general overview, we averaged the percentages of the positive elements on the Likert-scale per year per context to simplify the data and to make it easier to identify trends. The results of this average can be seen in Table 2. For the solo context, we can observe that the average steadily climbs by 3 percentage points each year from 57% to 60% to 63%. This shows a slight improvement in the perception of their importance. However, the group context starts at an average of 64% in 2021, then drops to 62% in 2022 and climbs to 69% in 2023 again. Even though the average dips by 2%pt. in 2022, this development is still positive.

Additionally, we averaged the software metrics excluding the testing related ones. Without these, we saw an improvement of 14%pt. between 2021 and 2023 in the solo context. The group context also shows a general, albeit less pronounced, improvement of 6%pt. between 2021 and 2023. Without the testing metrics, we do not see the previously observed small dip in 2022 and have a purely positive development in the three years.

Looking at the average of only the testing metrics, we can see the average improving from 63% to 67% in the solo context from 2021 to 2023. That same 4%pt. improvement from

73% to 77% can be seen in the group context as well, albeit with a higher starting number. Again, we have a small dip of 1%pt. and 6%pt. in 2022 for the solo and group contexts respectively.

For the statistical evaluation of the self-assessment data, we tested the hypothesis that the inclusion of TILE would improve the students' quality mindedness. To achieve this, we performed two simple linear regressions, both using the year as an independent (or predictor) variable. For the first linear regression, we looked at the effect of the year on the overall sumscore of the self assessment (dependent variable or outcome variable), "sumscore" for short. For the second linear regression, we focused on the six questions that we determined to be particularly closely related to testing: We examined the effect of the year on the sumscore of these questions. This variable we call "testing". If our hypothesis is correct and the effect is visible in the self-assessment, we would expect a positive slope in both simple regressions. For the simple linear regression, we followed the procedure detailed in [FMF12, chapter 7.4-5].

In a second step, we also explored whether the solo context vs. group context made a difference, either on its own or in interaction with the intervention, by using a multiple regression. We did not find that taking up these additional factors made a substantial contribution and are focusing on the simple linear regressions for the report.

		Estimate	Std. Error	<i>t</i> value	p-value (two-sided)	p-value (one-sided)
sumscore	(Intercept)	-3137.331	3919.282	-0.8	p=0.424	p=0.212
	year	1.585	1.938	0.818	p=0.415	p=0.207
testing	(Intercept)	-886.748	1611.891	-0.55	p=0.583	p=0.291
	year	0.450	0.797	0.565	p=0.573	p=0.286

Tab. 3: Results of the simple regressions

Table 3 contains an overview over the results of both of the simple linear regressions. The model predicting the overall sum score had a slope of 1.581: The overall score in the self assessment has grown by 1.59 points every year. The model predicting the testing related software metrics had a slope of 0.45: The score in the software metrics particularly related to testing has grown by 0.45 points every year. Because we had a directed hypothesis regarding the direction of the slope, i.e., that it should be positive because we expected an improvement over the years, the inferential statistical assessment can be based on the one-sided p-values. The effects of the year in both models were not statistically significant with one-sided p-values of  $p=0.21$  and  $p=0.29$ , respectively. However, as these are small effects, more participants would have likely been needed in order for them to become statistically significant (see [EGS13, p. 8.2]).

Table 4 shows an overview over the results regarding the assumption tests and plot<sup>5</sup> inspection. While some of the formal tests had p-values below the significance level of

<sup>5</sup> Find relevant plots over at [https://github.com/Alucard2112/TILE\\_and\\_MASS\\_Plots](https://github.com/Alucard2112/TILE_and_MASS_Plots)

assumption	statistic	sumscore			statistic	testing		
		p-value	visual	overall		p-value	visual	overall
autocorrelation homoscedasticity normal distribution	DW=1.717	p=0.043	+	+	DW=1.752	p=0.074	+	+
	$\chi^2=3.592$	p=0.058	+	+	$\chi^2=0.356$	p=0.550	+	+
	D=0.142	p=0.001	0	0	D=0.105	p=0.031	0	0

Tab. 4: Overview over assumptions for the simple linear regressions for both dependent variables

$\alpha=0.05$ , inspecting the relevant plots generally indicated an acceptable level of assumption violations. Additionally, while the Durbin-Watson (DW) test for possible autocorrelation violations also had *p-levels* below the significance level, the DW *test statistic* had levels close to the best possible score.

Regarding the assumption of normally distributed regression residuals, the picture is less clear. The Kolmogorov-Smirnov test was significant for both dependent variables. Inspecting the Q-Q plot showed that while most of the residual values are on a relatively straight line concordant with a normal distribution, in the tails there are some substantial deviations with some falling into the more relevant range between -2 and 2. Common linear transformations that can be used to mitigate the problem include taking the logarithm, the squareroot or the inverse of the dependent variable [FMF12, chapter 5.8]. For our case, neither of these transformations resulted in an improvement regarding the normal distribution assumption.

As the deviation in the untransformed data regarding the normal distribution assumption was mostly confined to the tails and the conclusions we are drawing from the data are limited, we concluded that the extent of the assumption violations was small enough to be acceptable.

In conclusion, the development in this part of the data is mostly positive. We have seen a positive development in the general importance of all nine software metrics in both contexts. Furthermore, the average of the six software metrics, those without the three pertaining to testing, is also positive. Excluding the six software metrics and only looking at the three that pertain to testing, we also saw a positive development, albeit with a small dip in the data set of 2022. To sum this development up, in all three aggregations (*overall*, *testing only* and *overall without testing*), we saw an improvement in our data.

### 2.3 Part III: Exam Data

For our second data collection action, we included a task in the final exams of the OoP lecture over the course of three years, which was also already outlined in the beginning of our study [DSB22]. In WiSe 2020 we've had 200 students taking the exam, 227 in WiSe 2021 and 185 in WiSe 2022. This task consisted of two sub-tasks: One focused on software

quality and the other on testing. In the first sub-task, the students were asked to fix a flaw within a given code-snippet. This ranged from missing documentation to bad performance or a missing input check. Within the second sub-task, the students were supposed to write a sufficient number of sufficient JUnit-tests to test the aforementioned code-snippet. The only thing we changed over the course of the three years was the code-snippet. We made sure that the snippet itself provided the same amount of fixable flaws.

	WS 20/21			WS 21/22			WS 22/23		
	Combined	Quality	Tests	Combined	Quality	Tests	Combined	Quality	Tests
Points	34%	41%	27%	38%	36%	40%	64%	59%	69%
ITC	$r=0.59$	$r=0.44$	$r=0.52$	$r=0.69$	$r=0.31$	$r=0.71$	$r=0.63$	$r=0.36$	$r=0.64$
Corrected ITC	$r=0.53$	$r=0.41$	$r=0.48$	$r=0.64$	$r=0.27$	$r=0.63$	$r=0.58$	$r=0.33$	$r=0.6$

Tab. 5: Item-Test-Correlation (ITC) and achieved points sorted by semester

Table 5 shows the achieved points, Item-Test-Correlation (ITC) and the Corrected Item-Test-Correlation (CITC) for the task and both sub-tasks. CITC is distinguished from ITC by subtracting the task itself from the overall achieved points in the correlation which makes it more accurate. Before the introduction of TILE and MASS, we saw a total of only 34% of achieved points in the task as a whole in WS 20/21. We also observed that students achieved more points in the quality focused task than in the testing one. This indicates that students had more trouble with testing than with quality in general.

Over the three years, we saw an improvement in the achieved points in both sub-tasks. Even though the quality task decreased by 5%pt. in WS 21/22, the testing task increased by a whopping 13%pt. which brings the overall score up to 38%. This is an increase of 4%pt. in comparison to WS 20/21 for the combined task. WS 22/23 saw another increase in both sub-tasks, landing at 64% for both. Especially the testing sub-task saw a huge increase from 27% to 69% over the three years. Because of TILE's focus on teaching testing, this positive trend is to be expected. The quality task also increased from 41% to 59% which is also a positive trend. The TILE approach should be the reason for this increase as it is based on test-driven development which increases general quality awareness in students [DJS08].

Both, ITC and CITC, saw a slight increase from 0.59 to 0.63 and 0.53 to 0.58 respectively over the three years with a small dip in WS 21/22. This can be explained in the achieved points for the combined task as they are stable between WS 20/21 and WS 21/22 and almost double in WS 22/23. If more points are achieved in this task and the overall points of students remain stable, then the correlation should worsen a little.

		Estimate	Std. Error	<i>t</i> value	p-value (two-sided)	p-value (one-sided)
sumscore	(Intercept)	13310.081	1780.333	7.476	$p<0.001$	$p<0.001$
	year	-6.563	0.881	-7.451	$p<0.001$	$p<0.001$
testing	(Intercept)	-33129.382	3025.488	-10.95	$p<0.001$	$p<0.001$
	year	16.42	1.497	10.97	$p<0.001$	$p<0.001$

Tab. 6: Results of the simple regressions for the exam results

For the statistical evaluation of the exam data, we looked at the effect of the year on the overall exam score as well as on the result in the diagnostic task. Like for the self-assessment data, we did so by performing two simple linear regressions using the year as independent or predictive variable and using the overall sumscore ("sumscore") and the score in the diagnostic tasks ("testing") as dependent variables, respectively. We transformed the variables "sumscore" and "testing" into percentages in relation to the best possible total score in order to compensate for differences between the exams regarding total scores.

Over the three observed years, we found a clear negative trend in the overall exam score: As Table 6 shows, with every year, sumscores were around 6.6 percentage points lower. This effect was statistically significant with a very low p-value at  $p < 0.001$ .

In contrast, for the diagnostic task, we found a clear positive trend. With every year, the score in the diagnostic task was around 16.4 percent higher. This effect was statistically significant as well with a very low p-value at  $p < 0.001$ .

For both simple linear regressions, we did not find any evidence for assumption violations with exception of the normal distribution assumption for the dependent variable "testing": The Kolmogorov-Smirnov test was significant with  $D = 0.103$  and  $p < 0.001$  and the QQ-plot gave indication for some relevant deviation from the normal distribution hypothesis as well. For this reason, we performed a bootstrapped version of the relevant simple linear regression following the procedure detailed in [FMF12, chapter 7.10]. The bootstrapped variant has the advantage of being robust against distribution assumption violations. The difference between the results for the classical and the bootstrapped version of the linear regression were negligible so we are focusing on the classical variant.

In conclusion, we saw an increase in student-proficiency of fixing quality flaws and writing tests. Furthermore, this did not bring down the ITC or CITC in any major way. Instead, we observed an overall ITC and CITC of above 0.4 which means that a good correlation exists [MK12, Chapter 4]. The statistical evaluation of the data also showed a strong significance within our data.

## 2.4 Part IV: Comparing Results

We were able to identify a positive trend in the subjective data concerning the overall attention paid. This specific positive trend is mimicked in the general achieved points in the combined task. Though it is a steady but slow increase in the subjective data, the change in the objective data was a steep jump. The only divergence we can find is the steepness of the improvement and not in the trend itself, which underlines the plausibility of our results.

Beginning with the software quality sub-task, we saw a slight decrease from 41% of achieved points to 36% from WS 20/21 to WS 21/22 before rising again to 59% in WS 22/23. Since the sub-task concerned itself with the quality of the code-snippet, specifically excluding tests, we use the average of the metrics without testing for a comparison. In that we saw the

average rising from 47 to 61 in the solo context and 60 to 66 in the group context. Both, the subjective and the objective results, show a positive trend. However, they behave differently over the three years as the subjective results do not mimic the dip in 2022 that we saw in the objective results.

Lastly, the sub-task that concerns itself with writing meaningful tests for the code snippet saw a positive trend from 27% in WS 20/21 69% in WS 22/23. Software metrics focused on testing also saw an increase from 63% in WS 20/21 to 67% in the solo context and 73% to 77% in WS 21/23. However, we observed the dip in 2022 again, albeit by only 1%pt. and 6%pt. respectively. Once more, both results share a positive trend, with the subjective results depicting that dip again.

Looking at the statistical evaluation, the analysed data tentatively indicates that the inclusion of the TILE approach did have a positive effect on the students' competence in testing specific tasks but not on the overall object-oriented programming competence. The self assessment data shows a small trend but as the effects are small and not statistically significant the self-assessment data alone is not sufficient for drawing conclusions regarding the hypotheses. In combination with the exam data, however, the trend becomes clearer. Particularly, the contrast between diagnostic task and overall score in the exam data is interesting: While the overall scores became statistically significantly worse over time, there was still a substantial and statistically significant positive trend for the testing-related task. At this point in time, we can not completely rule out that the, at the time, ongoing pandemic has had an effect on the students' performance as the later cohorts, specifically those in WS 21/22, were more or less affected by ongoing school closures and the online approach of the university.

### **3 Threats to Validity**

First of all, because of the nature of this research, we were not able to completely control our experiment. We've had outer influences that may have tainted our data like, of course, the pandemic. This affected all of our data collection.

Over the three years, the questions within the survey remained unchanged other than the inclusion of a comment field from year 2 onward. The only difference between the survey data collections for the three years is that we changed the participation in the survey from being optional in the first iteration to mandatory in the second and third iterations. This change caused the number of participants to skyrocket but might have led to students giving nonsensical answers. We adjusted conflicting answers by setting both of them to "did not know the metric". This interpretation of the data is quite conservative. This only applied to only 0.5%, 1.3% and 0.6% of the answers in the first, second and third year, respectively. As this discrepancy is not present in every metric and the percentage of answers affected is small enough, we attribute this to accidentally clicking on the wrong answer and not malicious intent.



There are a few data points of students who answered "Never heard of it" for each one of the self-assessment questions. It is possible that these results were produced by students who just wanted to quickly complete the survey and did not genuinely answer the questions. When excluding these data points, we found that the discovered effect almost vanishes. In retrospect, we've also identified a potential problem with the phrasing of the Likert-scale data points: "Never heard of it" can also be interpreted as a neutral answer instead of the worst possible one. This can have skewed the Likert-scale into a negative direction.

Regarding the statistical evaluation, the possibly violated normal distribution assumption is another threat to validity. While we considered the extent of the assumption violations for the self-assessment data acceptable, other more robust analyses could be explored for this part of the data in the future. Also some students repeated the course and participated in the survey more than once in self-assessment or/and exam, generating some overlap and probably some autocorrelation within the data. This could be examined with analyses that take into account the hierarchical structure of the data.

In addition, the assessment task was new in WS 20/21 whereas from WS 21/22 onward students may have practiced it when using old exams in their preparation.

## 4 Related Work

In our previous work [DSB22] we have already described the set-up of our experiment and the first set of data. But of course, only now we are able to analyze the development over the years. Another precursor to this paper is a joint publication on the TILE approach by Doorn et al. [Do23]. There we show a preliminary result of the TILE-approach within university teaching. In that paper we only looked at the diagnostic test over two years and did not analyze the data in a statistically rigorous way as in this paper.

Desai et al. [DJS08] conducted a meta study of experiments done at universities with teaching test-driven development (TDD) which TILE is based upon. They found that most studies found that TDD improves the students' quality awareness and increases their productivity. However, they found that an optimal point of introducing students to the concept has not yet been found as of the publication of their research.

## 5 Conclusion and Future Work

Over the three years that we have collected data, we found that including TILE and MASS has had a positive effect on our students' comprehending of software quality in both, their subjective view and in a more objective view. Specifically, their skill in writing correct and comprehensive unit-tests for a given software increased. Even though we could not find a statistically significant trend in the survey data (subjective view), we were able to find that the substantial effect in the diagnostic assessment *was* statistically significant. Therefore,

we conclude that the introduction of TILE and MASS has positively impacted the students' understanding of software quality and the importance of writing unit-tests for their code, even though they may underestimate their own quality understanding as is shown in the subjective data.

As for the future, we would need to continue the data-collection on TILE and MASS and improve upon their integration into the lecture. It would also be beneficial to add a field to the survey where students are able to share their own interpretation of the most negative point on the Likert-scale so it would be easier for future analyses to interpret their data.

## Acknowledgements

This work is partly funded by the Erasmus+ project *Quality-focussed Programming Education (QPED)*, 2020-1-NL01-KA203-064626.

## References

- [Co23] Cornillon, P.-A.; Hengartner, N.; Matzner-Løber, E.; Rouvière, L.: Régression avec R. EDP Sciences, Les Ulis Cedex A, 2023.
- [DJS08] Desai, C.; Janzen, D.; Savage, K.: A Survey of Evidence for Test-Driven Development in Academia. SIGCSE Bull. 40 (2), pp. 97–101, 2008.
- [Do23] Doorn, N.; Vos, T.; Marín, B.; Bockisch, C.; Dick, S.; Barendsen, E.: Domain TILes: Test Informed Learning with Examples from the Testing Domain. In: Research Challenges in Information Science: Information Science and the Connected World. Springer Nature Switzerland, Cham, pp. 501–508, 2023.
- [DSB22] Dick, S.; Schulz, S.; Bockisch, C.: A study on the quality mindedness of students, Software Engineering im Unterricht der Hochschulen (SEUH 2022), 2022.
- [EGS13] Eid, M.; Gollwitzer, M.; Schmitt, M.: Statistik und Forschungsmethoden. Beltz, Weinheim, 2013.
- [FMF12] Field, A.; Miles, J.; Field, Z.: Discovering Statistics Using R. SAGE publications, London, 2012.
- [FW11] Fox, J.; Weisberg, S.: An R Companion to Applied Regression. SAGE publications, Thousand Oaks, 2011.
- [MK12] Moosbrugger, H.; Kelava, A.: Testtheorie und Fragebogenkonstruktion. Springer-Verlag, Heidelberg, 2012.
- [VDM22] Vos, T.; Doorn, N.; Marín, B.: Test Informed Learning with Examples. In: CSERC '21. ACM Digital Library, pp. 1–2, 2022, ISBN: 978-1-4503-8576-3.

# SWTbahn: An Embedded Software Demonstrator in Symbiosis with Embedded Software Education

Bernhard M. Luedtke,<sup>1</sup> Eugene Yip,<sup>1</sup> Gerald Lüttgen<sup>1</sup>

**Abstract:** Demonstrators, such as model railways, are effective in providing real-world examples that help students to grasp abstract concepts in the engineering of embedded software. Although building such a demonstrator yourself can appear overwhelming, we argue how building your own improves your teaching of embedded software. We report on how we have achieved these improvements with a development process that lets our teaching co-evolve with our own demonstrator, share our main lessons learned, and invite educators to discuss remaining challenges.

**Keywords:** model railway demonstrator; evolutionary development; embedded software

## 1 Introduction

Students are exposed to many abstract concepts when learning about software engineering, but thinking abstractly can be a difficult competence for students to acquire. Students often struggle to see relevance in the taught concepts and paradigms, but grounding them with real-world examples can help keep students on track, motivated, and engaged [ALH07]. This is especially true for our Bachelor's module on *Reactive Systems Design (RSD)*, which covers the *model-driven engineering of embedded software* that has *real-time* and *safety-critical* concerns. Like other embedded software educators [Hö06; Mc07; Vö18], we have designed and built our own digital model railway, called *SWTbahn*, which serves as a practical demonstrator of embedded systems in our RSD module, projects, and theses, and offers students a realistic case study to put theory into practice. SWTbahn has supported 50% of our RSD practicals and tutorials, about 50% of our thesis topics over the last 6 years, and several semesters of 3 to 4 concurrent projects.

Having built SWTbahn, we take the position that building your own demonstrator improves teaching: (1) by being in control of the development process, you can set the foundation for an adaptable and extendable demonstrator that can keep up with developments in teaching; and (2) the development process generates real-world problems in the area of embedded software that make for great project and thesis topics for students. We elaborate on this in Sect. 2 and describe in Sect. 3 the development process we used to achieve these improvements. We share our main lessons learned in Sect. 4 and conclude this paper in Sect. 5 with remaining challenges that deserve an open discussion among software engineering educators.

---

<sup>1</sup> University of Bamberg, Software Technologies Research Group, Germany, [firstname.lastname@swt-bamberg.de](mailto:firstname.lastname@swt-bamberg.de)

## 2 How Building a Demonstrator Improves Teaching

While demonstrators can be obtained commercially, e. g. Lego Mindstorms<sup>2</sup>, we decided to build our own because we could see greater benefits to teaching even though the involved risk, investment, and complexity seemed higher. In the following, we describe two key ways in which building your own demonstrator can improve your teaching.

**Adapting to evolving teaching concepts.** As educators, we are always striving to improve our teaching, e. g. to explain concepts more intuitively, to adjust to changing student backgrounds, or to use more industry-relevant technologies. Although a demonstrator may initially be a valuable teaching aid, its value can diminish if its capabilities cannot be adapted to support changes in teaching concepts. This is often the case with commercial demonstrators that have been developed with a particular teaching concept and set of applications in mind. In contrast, when building your own demonstrator, you can control the technology stack by developing your own hardware and software solutions, tailor its capabilities to serve your current teaching concepts, and make it adaptable to future concepts. You also gain the know-how needed to make such adaptations. Moreover, an early prototype of the demonstrator can be used in teaching to test the demonstrator's effectiveness and to also refine the teaching itself. If it is not feasible for you to build your own demonstrator, you could try to find an open source alternative that can be adapted to support a wide variety of teaching concepts.

SWTbahn supports the concept-based learning approach of our RSD practical and tutorial sessions. Each session focusses on a selection of key concepts taught in the lectures, and aims to ground the theory on relevant areas of SWTbahn. For instance, students apply their knowledge of model-driven design and synchronous automata on exercises that guide them to model a train's physical behaviour as it reacts to drive and brake commands. To evolve our RSD module more towards research-oriented teaching, we have been able to adapt SWTbahn to showcase research in the areas of modeling, verification, deployment, and integration, and their practical relevance. Moreover, we were able to incorporate industry-relevant tools into our teaching by adapting SWTbahn to interoperate with *KIELER/SCCharts*<sup>3</sup> and *nuXmv*<sup>4</sup>.

**Offering real-world problems to students.** It can be difficult to create project and thesis topics that attract and motivate students, especially those that have outcomes with purpose beyond the work itself. We have found that, by building your own demonstrator, you regularly encounter challenges and topics that deserve further investigation. These you can refine into interesting real-world problem statements for students to analyze, design, and develop solutions for. Such problems appeal to students because they get exposed to challenges unique to embedded systems, become immersed in the engineering of capabilities for a real embedded system, and can see their work in a bigger picture. However, careful problem

---

<sup>2</sup> <https://www.lego.com/en-de/themes/mindstorms/learnprogram> (accessed 9 Nov 2023)

<sup>3</sup> <https://github.com/kieler> (accessed 9 Nov 2023)

<sup>4</sup> <https://nuxmv.fbk.eu> (accessed 9 Nov 2023)

selection is needed as not all may be appropriate for student work, e. g. when a timely resolution is needed, or when very specific expertise is required.

As an example, the development of SWTbahn’s initial software stack generated numerous topics, where some of the software is available via *GitHub*<sup>5</sup>. A Bachelor project student designed and developed a low-level C library for the model railway communication protocol *BiDiB*<sup>6</sup>, who was then hired to prototype a web application for the text-based remote control of SWTbahn<sup>7</sup>. Other projects have extended this with graphical user interfaces and real-time visualisations. Master’s and Bachelor’s theses have applied model-based testing to user workflows, studied the use of requirements management tools, prototyped a log message analyser to assist in troubleshooting and maintenance work, and developed a *domain-specific language* called *BahnDSL*<sup>8</sup> for the automated consistency checking and generation of configuration files and routing information. SWTbahn still offers many topics to students, e. g. the development of a digital twin, automated safety layer, verifiable models, algorithms for interlocking, train scheduling, and web applications for reactive systems.

### 3 Why Co-Evolving Your Teaching and Demonstrator is Beneficial

For the two improvements described in Sect. 2, the first relies on developing a demonstrator with adaptability in mind, and the second relies on integrating teaching (e. g. projects and theses) into the demonstrator’s development. Hence, the realization of these improvements relies on a development approach in which the teaching and the demonstrator can co-evolve. This section presents our development approach and Sect. 4 shares our lessons learned.

Our development approach can be summarized by a modified *Spiral model* [Bo86], shown in Fig. 1. The original Spiral model was conceived to guide a team through risky software development, which for us also included the risky construction of a model railway with no prior experience. Each time we cycled through the spiral, we aimed to co-evolve the teaching and demonstrator, and to produce more substantial artifacts. This approach helped us to navigate the breadth of possibilities, to engage students with early demonstrator prototypes, and to already improve our teaching before having completely built the demonstrator.

Each cycle through the spiral involves four phases: **(1) Brainstorm and develop teaching concepts** with the use of a demonstrator in mind. They can be refined by mocking up and testing teaching plans. Be creative and postpone feasibility concerns regarding the demonstrator to Phase 3. **(2) Elicit teaching-driven demonstrator requirements and constraints**, which will initially be vague, by analyzing the planned demonstrator usage in the developed teaching concepts. **(3) Design and develop the demonstrator** after carefully selecting an appropriate ecosystem of model railway hardware and software. Be

<sup>5</sup> <https://github.com/uniba-swt?q=swtbahn> (accessed 4 Jan 2024)

<sup>6</sup> <https://bidib.org/>; <https://github.com/uniba-swt/libbidib> (both accessed 4 Jan 2024)

<sup>7</sup> <https://github.com/uniba-swt/swtbahn-cli> (accessed 4 Jan 2024)

<sup>8</sup> <https://github.com/trinnguyen/bahndsl> (accessed 4 Jan 2024)

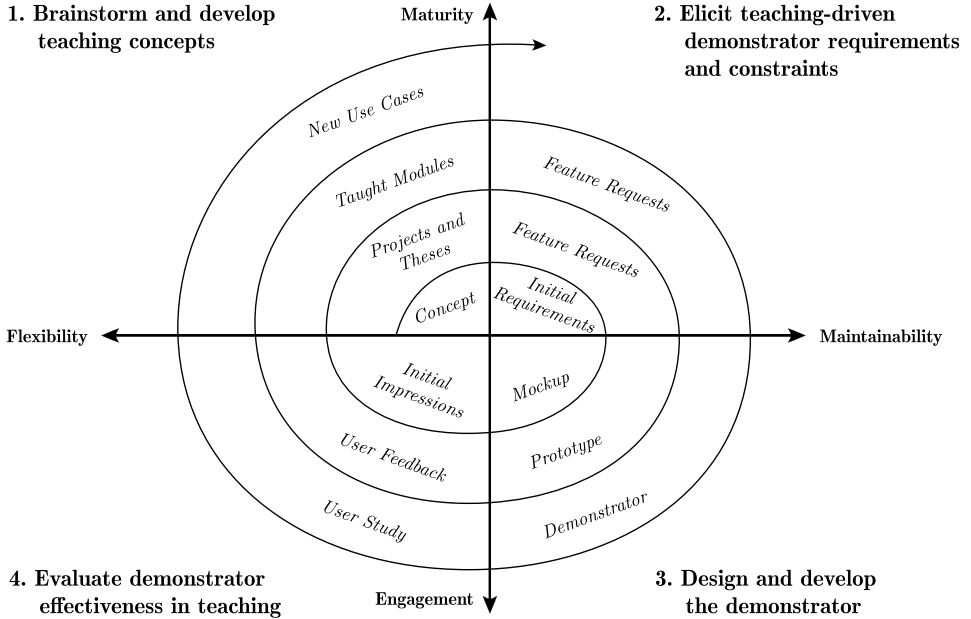


Fig. 1: Spiral model for developing a demonstrator alongside teaching. Example artifacts produced in each of the four phases appear inside the spiral. The axes are labelled with qualities to work towards.

more ambitious with each new prototype, but also aim to improve the construction quality. **(4) Evaluate demonstrator effectiveness in teaching.** Demonstrator prototypes can be phased into teaching to get early feedback on their effectiveness.

During the initial cycles, we established four desirable qualities for our teaching and demonstrator to possess: *maturity* where the demonstrator is an effective teaching aid, and teaching with the demonstrator is streamlined; *flexibility* where the demonstrator caters to a variety of teaching concepts and skill levels; *maintainability* where the demonstrator has documentation and personnel to keep it operational, and the teaching materials reflect the demonstrator's current capabilities; and *engagement* where many students, at different stages of study, work with the demonstrator. We considered our decisions in terms of their effect on the above qualities, as we wanted to strike a balance between the improvements we desired to achieve. For example, when adding capabilities to increase flexibility, we had to consider the increased maintenance effort. Similarly, when involving students in the SWTbahn software development process, we had to consider the quality of their code in relation to the reworking needed to achieve maintainability and maturity.

The Spiral model has been quite flexible in that we have been able to employ various development processes in the approximately 7 cycles since the conception of SWTbahn in 2016, e. g. *rapid prototyping* in earlier cycles, *Waterfall model* in later cycles, *bootstrapping*

across the cycles, and *agile development* to manage team momentum. Using these processes, we evolved from a small ( $1.3 \times 0.65$  m) single-loop prototype, that could only support one short train with a few branching tracks and only simple signals, to a larger ( $3.4 \times 0.9$  m) multi-loop track layout, that can support 5 long trains with many large clusters of branching tracks and a variety of railway signals. Students were hired to help construct and configure SWTbahn, from which they gained first-hand experience with embedded systems. The exact processes you employ depends on your team's expertise and experience and the ambitions of your envisioned teaching and demonstrator.

## 4 Lessons Learned

Our endeavor to co-evolve our teaching and demonstrator using the Spiral model has not always been easy. This section outlines the main lessons we have learned along the way.

**A disciplined work attitude is key to good maintenance.** The Spiral model, together with staff and students of various skills and expertise, does induce a heritage of artifacts and the resulting patchwork can be difficult to maintain. To manage this, we aim to be disciplined in the recording of decisions, documenting of findings and solutions, updating of employed technologies, and managing of student contributions. To ensure that artifacts and knowledge are retrievable and organized, we work in version-controlled repositories, use the included issue management and wiki, and perform code reviews to improve consistency and quality.

**Give students time to immerse themselves in demonstrator-related work.** Although students in our RSD practicals have enjoyed working with SWTbahn, they often struggle to complete relatively small exercises. This, however, is because students encounter many of the real challenges that one faces when designing software for embedded systems, e. g. how to communicate with the environment, handle concurrent events, satisfy real-time deadlines, ensure safety, and guarantee correctness. To not overwhelm students, each practical is designed to offer hands-on experience on specific concepts taught in the lectures, and we provide code skeletons to get students started. Our project and thesis students face similar challenges, but at a larger scale. Instead of developing small isolated software components, they are expected to develop substantial software solutions that interface with one or more SWTbahn subsystems. We thus scope the problem statements such that students have the time to familiarize themselves with SWTbahn and its associated technologies and domains.

**Expect the development team to also evolve.** Being in a university environment, we experience high staff and student turnover. This often disrupts the development momentum and places maintainability at risk. To mitigate the disruption in momentum, we hold weekly meetings with the purpose of assessing our progress and priorities. This has allowed us to adapt to team changes with smoother handovers, onboarding, and redistribution of work when members become busy with other commitments. Furthermore, we always try to assign two long-term staff members to the management of all SWTbahn-related activities to ensure continuity when one of them leaves.

## 5 Conclusions

Building our model railway demonstrator, SWTbahn, has been a long-term commitment, but as a result we have been able to improve our teaching of embedded systems. We continue to evolve SWTbahn with our teaching concepts, and SWTbahn continues to offer real-world problem statements for students to work on. We have received positive feedback from our students, who have enjoyed using SWTbahn to apply concepts of embedded software design and development. Our modified Spiral model has greatly facilitated our co-evolutionary approach to developing our teaching and demonstrator together, and we would recommend it to others who are interested in building their own demonstrator. However, several challenges remain for which we offer the following questions to the community of software engineering educators to start an open discussion:

- (1) How can practicals and projects, which involve a complex demonstrator, be designed to accommodate a large number of participants?
- (2) What does the end-of-life of a complex demonstrator look like? When does it make sense to phase out a demonstrator?
- (3) With artificial intelligence (AI) powered tools and the embedding of AI into safety-critical systems becoming more widespread, how can a physical demonstrator support the teaching of these topics?

**Acknowledgments.** We thank the reviewers and our colleagues Jan H. Boockmann and Kerstin Jacob for their suggestions that have helped us to improve the paper.

## References

- [ALH07] Arnold, R.; Langheinrich, M.; Hartmann, W.: InfoTraffic: Teaching Important Concepts of Computer Science and Math through Real-World Examples. In: SIGCSE. ACM, pp. 105–109, 2007.
- [Bo86] Boehm, B. W.: A Spiral Model of Software Development and Enhancement. SIGSOFT Software Engineering Notes 11/4, pp. 14–24, Aug. 1986.
- [Hö06] Höhrmann, S.; Fuhrmann, H.; Prochnow, S.; von Hanxleden, R.: A versatile demonstrator for distributed real-time systems: Using a model-railway in education. In: Workshop on Dependable Embedded Systems. ERCIM, 2006.
- [Mc07] McCormick, J. W.: Model Railroading and Computer Fundamentals. Computer Science Education 17/2, pp. 129–139, 2007.
- [Vö18] Vörös, A.; Búr, M.; Ráth, I.; Horváth, Á.; Micskei, Z.; Balogh, L.; Hegyi, B.; Horváth, B.; Mázló, Z.; Varró, D.: MoDeS3: Model-Based Demonstrator for Smart and Safe Cyber-Physical Systems. In: NASA Formal Methods. Springer, pp. 460–467, 2018.



## Session 3



# Was heisst “Programmieren” im Zeitalter von LLM-basierten Programmier-Assistenten?

## *Positionsbeitrag*

Thomas R. Gross<sup>1</sup>

**Abstract:** Auf “Large Language Models” (LLMs) basierende Programmier-Assistenten, wie z.B. GitHub Copilot, AWS Code Whisperer, oder Google Bard stellen die (universitäre) Lehre der Informatik vor neue Herausforderungen. Wenn wir jetzt das “End of Programming” erreicht haben, warum sollen dann Studierende noch Programmieren lernen (und Dozierende dieses Fach unterrichten)? Während verschiedene Beiträge die Reaktionen der Informatik Dozierenden dokumentieren (und die Auswirkungen solcher Werkzeuge auf formative und summative Evaluation untersuchen), so ist auch eine Diskussion über die inhaltlichen Folgen dieser Werkzeuge nötig. In diesem Beitrag argumentiere ich, dass bei Einsatz dieser Werkzeuge (i) Programmieren weiterhin notwendig für die Ausbildung in Software Engineering (und anderen Gebieten der Informatik) ist, und (ii) Programmieren als Grundlage für die Entwicklung von Software Systemen weiterhin relevant ist und als solche unterrichtet werden sollte.

**Keywords:** LLM-basierte Werkzeuge; Programmieren; Software Technology

## 1 Einleitung

In den letzten 2 - 3 Jahren sind Programmier-Assistenten, die auf Large Language Models (LLMs) basieren, so weit fortgeschritten, dass sie in gängige Entwicklungsumgebungen (IDEs) integriert werden konnten und damit das Codieren in vielfacher Hinsicht erleichtern. Basierend auf einer grossen Anzahl früher geschriebener Programme können diese Werkzeuge in vielen Situationen Vorschläge machen, und die Benutzer/innen solcher Werkzeuge geben dann Hinweise (oder weitere Anforderungen) ein, um das erwünschte Programm zu erstellen. Beispiele solcher LLM-basierten Werkzeuge sind GitHub Copilot, AWS Code Whisperer, Google Bard oder Meta Code Llama.

Im folgenden beziehe ich mir (nur) auf GitHub Copilot, dass nach den Beobachtungen anderer Dozierender durch verschiedene Ansammlungen von Informatik Programmierproblemen trainiert wurde[Be22]. Insbesondere passen “klassische” Programmierprobleme (die in vielen Büchern zur Einführung in die Programmierung verwendet werden) gut zum Lösungsansatz solcher Werkzeuge, und verschiedene Projekte haben die Fähigkeiten dieser Werkzeuge dokumentiert [DKG23, We23b, BJP23]. Prüfungen in einem solchen Umfeld

---

<sup>1</sup> ETH Zürich, Departement Informatik, 8092 Zürich, Schweiz

sind daher eine Herausforderung für universitäre Programmierkurse[CA23, Ou23] und die Dozierenden verschiedener Institutionen haben unterschiedlichste Strategien zur Integration (oder dem Verbot) dieser Werkzeuge vorgeschlagen[LG23].

Allerdings wird durch die Programmier-Assistenz eine Lösung oft nicht sofort (nach Eingabe der Anforderungen, d.h. der zu lösenden Aufgabe) erstellt, sondern die Benutzer/innen müssen die Programmier-Assistenz anleiten, eine vollständige richtige Lösung zu erstellen<sup>2</sup>. Im Raum steht die Behauptung, dass diese Werkzeuge das “End of Programming” zur Folge haben [We23a] und Programmierkenntnisse für zukünftige (und jetzige) Informatiker nicht mehr relevant sind. Niemand wird Software warten müssen – wenn eine neue Herausforderung identifiziert ist (oder ein neuer Bug entdeckt wurde), dann wird das Programm neu mit einem Werkzeug generiert (und nicht durch Modifikation des bestehenden Programms realisiert).

## 2 Das “End of Programming”?

LLM-basierte Werkzeuge nutzen aus, dass es für ein gegebenes Problem (oder zumindest für ein sehr ähnliches Problem) bereits (Teil)Lösungen gibt, die dann zu einer Lösung für das gegebene Problem kombiniert werden können. Für viele der Probleme, die in Programmierkursen verwendet werden, sind natürlich die Lösungen bekannt, und so haben diese Werkzeuge keine grosse Mühe, eine Lösung zu finden. Aber für viele reale Projekte ist eine grosse Code Basis nicht vorhanden, so dass es viel zu früh ist, vom “End of Programming” zu sprechen[Ye23].

Auch unsere Erfahrung mit GitHub Copilot unterstützt die Einschätzung, dass der Nachruf auf Programmieren verfrüht ist. Tabelle 1 zeigt wieweit Copilot die wöchentlichen Prüfungsaufgaben eines Semesters einer “Einführung in die Programmierung” Vorlesung (die Java als Programmiersprache verwendet) für das 1. Semester lösen konnte. Die 1. Spalte gibt an, in welcher Semesterwoche ein Problem gestellt wurde. Die 2. Spalte gibt das Thema der Aufgabe an. Keine dieser Aufgaben konnte Copilot direkt *ohne weiteren Input* (d.h. ohne Hinweisungen und Anweisungen) lösen. Spalte 3 zeigt an, welchen Prozentsatz der Unit Tests das mit Hilfe von Copilot innerhalb nützlicher Zeit erstellte Programm korrekt ausführen konnte<sup>3</sup>.

Es ist nicht überraschend, dass Copilot mit Hinweisen und Anweisungen in der Lage ist, die Aufgaben zu lösen. Aber die Hinweise und Anweisungen setzen profunde Programmierkenntnisse voraus. Insbesondere mussten die Benutzer/innen Hinweise zur Wahl

---

<sup>2</sup> Diese Einschätzung wird auch von den Entwicklern Werkzeuge geteilt; GitHub Copilot wird als “Your AI *Pair Programmer*” beworben.

<sup>3</sup> Die Programmieraufgaben wurden auf Deutsch gestellt. Frühere Experimente (mit den Sprachen Deutsch und Italienisch) bestätigten, dass die Wahl einer (gängigen) Sprache für die Aufgabenstellung nicht relevant ist. Die automatische Spracherkennung und anschliessende Übersetzung sind gut genug für die LLM-basierte Programmierassistentz.

Woche	Prozentsatz Unit Tests	
W4	Arrays, control flow	100
W5	Data structures, control flow	100
W6	Arrays, recursion	66.67
W7	Graph manipulation, references	100
W8	Paths, references, recursion	100
W9	Objects, simulation, inheritance	100
W10	Collection Framework , exceptions	100
W11	Collection Framework, object design	100
W12	Maps, object design, overriding	32.58

Tab. 1: Bearbeitungserfolg von Copilot für verschiedene Prüfungsaufgaben.

der Datenstrukturen (bzw. Darstellung) geben. Diese Anforderung kann natürlich auch eine Nebenwirkung der Aufgabenstellungen sein, die die Verwendung des Java Collection Frameworks betonen.

Wenn also Programmierer/innen mit diesen Werkzeugen erfolgreich arbeiten wollen, dann müssen sie weiterhin Programmierkenntnisse haben – und da Copilot das volle Spektrum der Programmiersprachen (in unserem Fall: Java) nutzt, müssen Programmier/innen mit allen Konzepten (insbesondere des Objektsystems) vertraut sein. Diese Kenntnisse sind auch erforderlich, wenn man den Output von Copilot überprüfen möchte, z.B., um festzustellen, dass das erstellte Programm nicht unerwünschterweise Informationen weitergibt (wobei es keine Rolle spielt, ob solches Verletzen der Erwartungen absichtlich oder unabsichtlich gemacht wird).

### 3 Was heisst “Programmieren”?

Der Einsatz von Programmier-Assistenten wie Copilot kann Programmieren interessanter machen, da die Assistenz manche wichtige aber letztlich unkritische Aufgabe übernimmt. Wenn man für eine gegebene Klasse den Konstruktor finden muss, der am zweckmässigsten die Objektexemplare initialisiert, dann macht Copilot oft einen guten Vorschlag (und ist weniger mühsam als aus den Optionen der Code Completion allein (oder gar der API) die beste Auswahl zu treffen). Copilot kann also eine grosse Hilfe für den Schritt des Codierens sein. Aber wenn wir unter “Programmieren” (auch) das Zerlegen eines Problems in Teil-Probleme und das Zusammenfügen der (Teil)-Antworten zur Lösung eines Problems verstehen [Ce16] dann hilft Copilot aber löst nicht das Problem des “Programmierens”.

Copilot ist auch nur begrenzt hilfreich im kritischen Lesen der Aufgabenstellungen. Eine Programmier-Assistenz kann helfen, verschiedene Varianten (eines Programms zu erstellen), aber welche der Varianten denn die Anforderungen am besten erfüllt (und welche Aspekte in der Aufgabenstellung noch festgelegt werden müssen, um das Verhalten genügend genau

zu spezifizieren) ist eine Frage, für die ein Programmgenerator nur bedingt geeignet ist. Es wäre daher wünschenswert wenn Veranstaltungen, die das Programmieren lehren, verstärkt diese Aspekte der Software Technologie betonen um Studierende auf eine produktive Arbeit mit Programmier-Assistenzen vorzubereiten.

## 4 Bemerkungen

LLM-basierte Programmier-Assistenten bieten eine Möglichkeit, das Programmieren interessanter zu machen. Aber die Interaktion mit solchen Werkzeugen setzt noch immer voraus, dass die Benutzer/innen programmieren können. Programmieren ist daher weiterhin notwendig für die Ausbildung in Informatik und insbesondere in Software Engineering. Ein wichtiger Teil des “Programmierens” ist das Erstellen (und Überprüfen) von Anforderungen. Programmieren ist eine Grundlage für den Entwurf und die Entwicklung von Software Systemen und die Ausbildung sollte diesem Umstand Rechnung tragen.

## Literaturverzeichnis

- [Be22] Berger, E.: , Coping with Copilot. Blog, Aug 2022.
- [BJP23] Barke, Shraddha; James, Michael B.; Polikarpova, Nadia: Grounded Copilot: How Programmers Interact with Code-Generating Models. Proc. ACM Program. Lang., 7(OOPSLA1):85–111, 2023.
- [CA23] Cipriano, Bruno Pereira; Alves, Pedro: GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2023, Association for Computing Machinery, New York, NY, USA, S. 61–67, 2023.
- [Ce16] Cerf, Vinton G.: Computer Science in the Curriculum. Commun. ACM, 59(3):7, feb 2016.
- [DKG23] Denny, Paul; Kumar, Viraj; Giacaman, Nasser: Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, S. 1136–1142, 2023.
- [LG23] Lau, Sam; Guo, Philip: From "Ban It Till We Understand It"to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In: Proc. ICER'23. ACM, August 2023.
- [Ou23] Ouh, Eng Lieh; Gan, Benjamin Kok Siew; Jin Shim, Kyong; Wlodkowski, Swavek: ChatGPT, Can You Generate Solutions for My Coding Exercises? An Evaluation on Its Effectiveness in an Undergraduate Java Programming Course. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2023, Association for Computing Machinery, New York, NY, USA, S. 54–60, 2023.
- [We23a] Welsh, Matt: The End of Programming. Commun. ACM, 66(1):34–35, 2023.

- 
- [We23b] Wermelinger, Michel: Using GitHub Copilot to Solve Simple Programming Problems. In (Doyle, Maureen; Stephenson, Ben; Dorn, Brian; Soh, Leen-Kiat; Battestilli, Lina, Hrsg.): Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 1, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023. ACM, S. 172–178, 2023.
- [Ye23] Yellin, Daniel M.: The Premature Obituary of Programming. Commun. ACM, 66(2):41–44, 2023.





## Session 4



# A Conceptual Framework to Transform Coding Education in Times of Generative AI

Stefan Bente,<sup>1</sup> Natasha Randall,<sup>2</sup> Dennis Wäckerle<sup>3</sup>

**Abstract:** In light of the rising ubiquity and accessibility of AI tools, software engineering curricula must be adapted in response, particularly with regards to coding education. We propose an extension to Bloom’s Revised Taxonomy, enhanced to a second dimension that represents the scope and complexity of coding tasks. The path that a computer science study program typically follows can be mapped onto these dimensions, and evaluated in the context of generative AI tool use, which falls into three core stages. AI should be initially banned, then made an explicit part of coding education, and finally becomes an implicit part of the coding practices of students.

**Keywords:** Coding; ChatGPT; Software Engineering; Education; Bloom’s Revised Taxonomy

## 1 Introduction

The rise of accessible generative AI tools such as ChatGPT and GitHub Copilot - capable of producing large amounts of text and code at the click of a button - has led to fears that students will simply use AI to cheat on assignments, at the expense of learning [SKM23]. These fears have been validated by research from Geng et al. suggesting that ChatGPT is capable of passing university level programming courses [Ge23b] [Ma23]. On the other hand, Krüger and Gref demonstrate how ChatGPT’s mathematical limitations can significantly reduce its effectiveness in a computer science degree program [KG23], and a new study by Jimenez et al. reveals how ChatGPT completely fails to handle real world software engineering tasks [Ji23]. Nevertheless, many developers increasingly believe that traditional programming is “headed for extinction” [We22].

A considered approach to generative AI in educational settings will therefore be key. But how exactly should SE curricula be adapted, specifically with regards to coding education, i.e. teaching programming skills? There are a number of recent publications on this topic ([Ab23], [Li23], [DB23], [Ja23], or [Ge23a], just to name a few). However, what is lacking is an easy-to-understand mental model “where, when, and how” to integrate generative AI into an SE curriculum. In section 2 we attempt to close this gap by proposing a simple framework to assess coding education. We enhance Bloom’s Revised Taxonomy [AKB01] to a second dimension, namely the “scope” of coding tasks, which encapsulates aspects

---

<sup>1</sup> TH Köln, Fakultät für Informatik und Ingenieurwissenschaften, stefan.bente@th-koeln.de

<sup>2</sup> TH Köln, Fakultät für Informations- und Kommunikationswissenschaften, natasha.randall.uni@gmail.com

<sup>3</sup> TH Köln, Fakultät für Informatik und Ingenieurwissenschaften, dennis.waeckerle@smail.th-koeln.de

relating to the complexity of a task. In section 3 we then validate this conceptual framework by analysing the Computer Science Bachelor at TH Köln, with a study that examines the extent to which ChatGPT is able to solve curricular assignments focusing on coding skills.

The combination of these two dimensions allows us to draw the “path” a computer science study program typically follows. This path can be segmented into three stages regarding how to embrace AI tools in coding, as described in section 4. Initially, we recommend to *ban AI* in the very first stages of programming education, to allow students to develop basic hands-on coding knowledge. In the second stage, we propose to make the use of AI tools an *explicit part of the coding education*, to let students develop a sense for the strengths and weaknesses of AI tools. Finally, in that last stage, AI tools should be an *implicit, unconstrained part of coding practice*, even up to their availability in exams - just the situation the students will face later in their professional context. Finally, we discuss open issues and further research of our proposal in section 5.

## 2 A Conceptual Framework to Describe Coding Education

A fundamental concept underlying effective coding education is *Active Learning* [HRL20, p. 23ff], which simplifies to the idea that true understanding comes through hands-on experience. As novices gradually progress, they ascend to higher cognitive levels, such as analyzing requirements and domains, and assessing existing code and technologies. Bloom’s Revised Taxonomy [AKB01, p. 28f.] provides a framework for understanding such levels, encompassing six distinct stages in a “somewhat hierarchical”<sup>4</sup> framework, ranging from lower-order thinking skills like remembering and understanding to higher-order skills such as applying, analyzing, evaluating, and creating. It provides educators with a structure to design learning activities and assessments that promote progressively deeper cognitive engagement.

For beginners in coding, the journey typically begins with small code snippets, aligning with the lower cognitive levels in Bloom’s Taxonomy: remembering, understanding, and applying concepts. But how to progress from here? The reality as an IT professional means dealing with large software systems, with up to millions of Lines of Code (LoC), and a multitude of technologies, architecture styles, and code patterns. This is a long way from the 10-100 LoC coding tasks for novices. This journey requires *two dimensions* to be described properly: On the one side, as students advance, they should be able to analyse requirements to turn them into code (Bloom level 4), read, understand, and assess the quality of code (level 5), and even think of new, innovative coding patterns or technologies (level 6). This dimension of students’ learning journey through IT problem solving is well covered by the Bloom taxonomy.

---

<sup>4</sup> The question if the Revised Bloom’s Taxonomy forms a *cumulative hierarchy* has more to it than meets the eye. The authors themselves give a differentiated “yes-and-no” answer that is worth reading [AKB01, p. 267f.].

On the other side, students' coding assignments should evolve to encompass the development of full-fledged programs with 1000 - 10.000 LoC. Structural considerations like software architecture become more and more relevant on the way: A combination of perfectly well designed small code snippets may ultimately end up becoming a horrifying "big ball of mud", if such aspects are disregarded. This dimension of "learning to code" is not covered by Bloom's Revised Taxonomy<sup>5</sup>. We therefore propose a second dimension that we call *scope*. Scope encompasses various aspects that collectively contribute to a task's complexity, including (but not limited to) size, number of classes, technology layers (like persistence annotations for an ORM layer), architecture styles (layered architecture, hexagonal, ...), Clean Code rules, code quality metrics (number of dependency cycles, cyclomatic complexity, ...), performance metrics, etc. For the sake of simplicity, within the context of this paper, we assume that the average size of the solutions' code base can serve as a valid indicator for scope, and that larger coding exercises will (have to) cover these additional aspects to provide a useful learning experience to students.<sup>6</sup>

Figure 1 depicts these two dimensions. The coding-related modules of an SE curriculum need to follow a structured path from the bottom left corner (low cognitive level, small code snippets) to intricate programming challenges with "Evaluate" level and large software systems in the top right corner. The stages that students traverse in their curriculum appear as quadrants in a 2-dimensional matrix.

Initially, the students need to *master the basics of coding*, mainly by applying (Bloom level 3) programming language syntax rules to small tasks (bottom-left). In the second stage, the students move on to *unassisted coding* (top-left), with the ability to analyze (level 4) requirements and domains, and evaluate (level 5) existing code for refactoring, to optimize it for quality attributes like maintainability or performance. In the last stage (top-right), students will slowly broaden the scope and use their skills to *develop complex solutions*. The bottom-right quadrant has no educational value, as it would just mean applying the same detailed instructions over and over to a very large task, without ever gaining a deeper understanding.

The y-axis in figure 1 employs decimal values for Bloom levels, not discrete numbers. If a

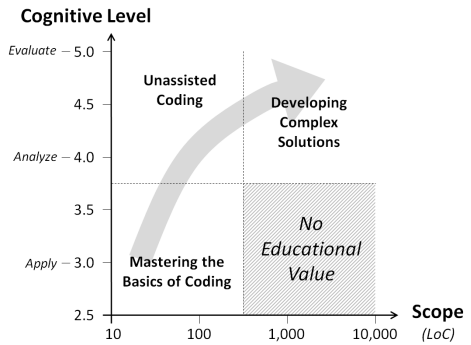


Fig. 1: Coding Education as a 2-dim Matrix

<sup>5</sup> Anderson and Krathwohl do actually propose a second dimension of their taxonomy, called the "knowledge dimension" [AKB01, p. 28ff.]. It distinguishes between factual, conceptual, procedural, and meta-cognitive knowledge. This is of course a relevant distinction, but doesn't really help us here in describing how coding exercises need to evolve through the curriculum.

<sup>6</sup> This assumption will require further research, see also the outlook in section 5 of this paper.

coding-related module's practical e.g. consists of 20 assignments, we assessed the highest required Bloom level for each of them, then calculated the average, resulting in a decimal number. Likewise, the LoC is determined by averaging assignment LoC. Data points were generated using either existing student solutions or newly created AI solutions. The exact data is detailed in table 1. In addition, the range of values in the chart is deliberately restricted between 2.5 and 5. The underlying assumption is that no module teaching coding (or anything else, for that matter) will stay mainly on Bloom level 1 or 2 - this would mean that students only learn by heart or explain concepts, but never actually *apply* any of the competences they acquire. A Bloom level higher than 5, on the other hand, will be found in the realm of a Bachelor thesis, but rarely (as an average over the whole semester) in a regular course. The evaluation of a concrete study program, as described in the section 3, supports this assumption.

ID	Module Name	Sem.	Description	Coding ECTS	Avg. Bloom	Avg. LoC
AP1	Algorithms and Programming I	1	Programming exercises in C and Java	8	3.2	35
AP2	Algorithms and Programming II	2	Programming exercises and patterns in Kotlin	7	3.5	117
ALG	Algorithmics	3	Algorithms (sorting, graphs, ...)	1.7	4.0	263
DB1	Databases I	3	ERM and basic SQL	2.3	3.4	43
PoP	Paradigms of Programming	3	OO vs. functional vs. logic programming	5	3.4	53
SE1	Software Engineering I	3	Domain modelling in Java and Spring	1.6	3.8	51
DB2	Databases II	4	Advanced SQL in client-server systems	5	4.0	1507
SE2	Software Engineering II	4	Complex system according to DDD in Java	4.1	4.4	1937
AI	Artificial Intelligence	5	Practical AI application	5	4.7	653
CSP*	Computer Science Project	5	Practical programming project	10*	>=4.0*	>= 5000*
PP*	Practical Project	6	Project intended as base for BA thesis	15*	>=5.0*	>= 10000*
BA*	Bachelor Thesis	6	Scientific thesis, sometimes on coding topics	12*	>=5.0*	>= 10000*

Tab. 1: (Potentially) Coding-related Modules in Computer Science Bachelor at TH Köln

### 3 A Brief Case Study: Computer Science Bachelor at TH Köln

In sample a case study, we analyzed the Computer Science Bachelor program at TH Köln. The coding-related parts of this program amount to 47 - 75 ECTS<sup>7</sup>.

We evaluated the mandatory coding-related courses regarding their average code size and Bloom level; the results are shown in table 1 (German module names have been translated to English). The numbers in table 1 were obtained by analyzing sample student solutions, or reference solutions provided by the supervisors, where available<sup>8</sup>. The modules marked with an asterisk are only potentially coding-related. Their numbers represent an

“optimistic” value obtained by averaging selected samples with a high degree of coding. Figure 2 shows the modules in table 1 positioned in the 2-dimensional Bloom grid introduced in section 2, marked by their ID and semester in which they should take place according to the study plan. The potentially coding-related modules are depicted as dotted circles, to visualize the high degree of uncertainty.

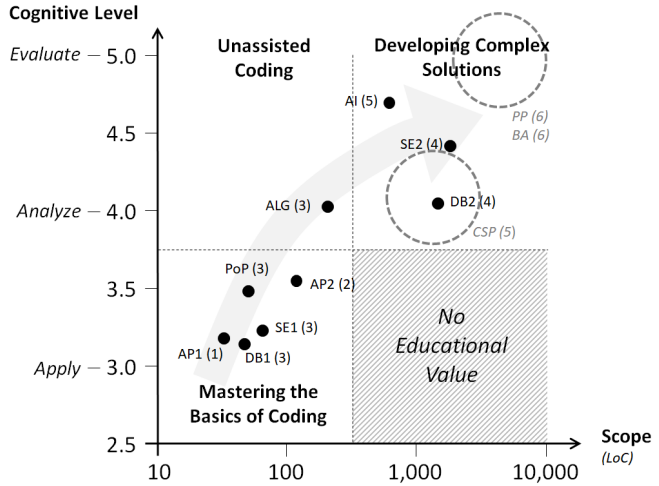


Fig. 2: Positioning of Modules in 2-dim Bloom Matrix

#### 3.1 ChatGPT and Current Assignments

To ascertain the impact of AI tools on coding exercises, we attempted to solve a subset of the modules’ assignments in table 1 (AP1, PoP, and SE2) using ChatGPT 3.5 and 4.0. We used an *uneducated approach*, meaning that only the tasks’ descriptions, outputs generated by the compiler, test outputs, and bugs could be provided to ChatGPT. No other input or manual coding was allowed.

<sup>7</sup> Elective courses, Bachelor thesis, and various project formats can contain a varying amount of coding-related content, hence the range.

<sup>8</sup> The way LoC are counted is by looking at the *context* in which the exercise happens.

In AP1 and PoP, for instance, there is a new (relatively small) context in every exercise, therefore each context is relatively small. In AP2 students sometimes have to extend the work from a previous exercise. In SE2, iteration on the same code base is a core principle, therefore the LoC numbers are higher.

The modules that we mainly focused on during this research were AP1 and SE2. AP1 is an introductory module, teaching students the very fundamentals of programming in C and Java. It is a mix of procedural and object-oriented assignments, completed throughout the entire semester. All of these tasks are elementary, do not involve any complex maths or logic, and most can be completed in a single file or class.

SE2 on the other hand, is a more advanced module teaching students how to develop larger scale applications using Domain-Driven Design (DDD) and Spring Boot. SE2 has fewer but larger exercises in Java, designed as milestones that all iterate on the same application. The tasks are free-form, allowing the students to build the application in any way they see fit, as long as it implements the necessary interfaces and complies with the regression and architectural tests. These applications always span multiple classes which are in the double digits, and thus have an entirely different scope than the coding exercises in the other modules. The complexity and the free-form approach requires a higher level on the Bloom Taxonomy, with most tasks being level 4 and some even reaching level 5.

It was immediately obvious that generative AI tools excel in generating programs small in scope. AP1 (1st semester) could essentially be solved in its entirety with no input other than the task description, within one hour. The same result was achieved with PoP (3rd semester), where ChatGPT was able to solve the given problems mostly on the first try.

Looking at SE2, when attempting to solve the milestone 0 of the summer semester of 23, which required the implementation of a simple B2B webshop, the entire milestone could be solved using no human written code, despite the large scope. However, the AI started struggling in the next milestone once architectural best practices started being enforced, which the AI seemingly was unable to solve on its own, especially regarding resolving cyclic dependencies. Furthermore, some of its default implementations were violating DDD principles, which would cause the code to fail tests in later milestones.

Meanwhile, the AI could not solve milestone 0 of the summer semester of 21, where a program should be implemented which simulates the movement of robots on a grid of cells containing barriers between some cells. The AI was incapable of implementing the necessary logic for the barriers, meaning that it was failing the automated tests, despite the scope of the task being slightly smaller than the tasks of the summer semester of 23. While this result seems surprising at first sight, it also confirms the AI tool's well-known struggle with maths and similar algorithmic challenges [Zh23].

Our observations match the results from numerous other studies on ChatGPT and curricular coding assignments. "Our findings show that ChatGPT is effective for easy and popular coding problems but is less reliable for harder and less popular problems" write Kuhail et al. [Ku23]; [KG23], [DB23], and [Ge23a] come to a similar conclusion. Single file programs with no additional challenges can easily be solved by students using an "uneducated" approach with ChatGPT, i.e. without ever trying to understand the assignment. The capabilities mentioned above are further enhanced when using widely known technologies



and programming languages. With more complex assignments, in the top-right *Develop Complex solutions* quadrant, ChatGPT can still be an extremely helpful tool, but nevertheless requires some manual interventions. Students will have a hard time solving such assignments with a purely uneducated approach.

## 4 Recommendations Regarding AI Tool Usage

Based on the capabilities of AI tools, new ways of teaching and learning must be developed. Jacques [Ja23] states that “[...] at the very least, we have a responsibility to prepare our students for this transition. Specifically, we need to consider how to develop good programmers while still acknowledging and engaging with these new tools for programming.”

Combining the conceptual grid from section 2 and the results from our case study in section 3 (and many others), we propose three stages in dealing with AI in SE curricula. This is depicted in figure 3.

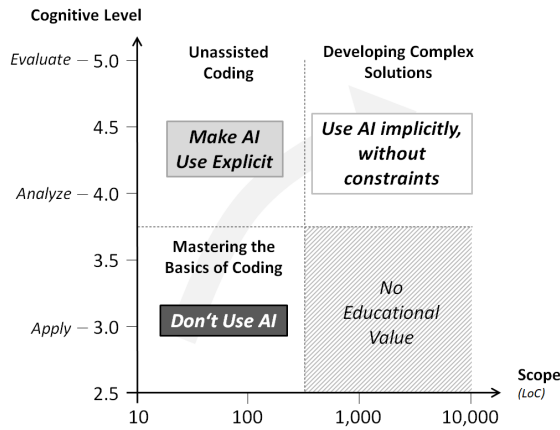


Fig. 3: Three Stages for AI Usage in SE Curricula

### 4.1 Stage 1 (Mastering the Basics of Coding): Don't Use AI

In the first stage of learning to code, it seems advisable to ban the use of AI tools during exercises, practicals, and exams. In this phase, students learn by simply applying syntax definitions, and by following quite detailed instructions. AI tools like ChatGPT offer an all too easy way out of this, as demonstrated in section 3.1. In addition, an unpublished research paper [DOH22] from the 2022 “Coding Excellence” course in Master Digital Sciences showed that beginners actually profit more from simpler programming tools, like syntactical code completion - rather than full-fledged AI tools like ChatGPT, GitHub Copilot, Tabnine etc., which are seemingly prone to creating a cognitive overload for novices.

Exams in coding-related courses should be digital and based on real-life development environments, if possible (following the “Constructive Alignment” principle [Bi03]). If the university has a large enough computer lab, then exams should take place there, as it is relatively easy to block access to unwanted tools in such a controlled environment.

## 4.2 Stage 2 (Unassisted Coding): Make AI Use Explicit

In this second stage of their “coding journey”, students have acquired the fundamental knowledge about how to write code - from a *computational thinking* approach to problem solving (at least for isolated tasks) to a basic familiarity with the syntax of one or two main programming languages, and they are capable of developing small scale programs on their own. This stage therefore focuses on deepening their understanding of the numerous aspects associated with the coding lifecycle, like domain exploration, algorithms, data structures, code patterns, quality attributes (e.g. through established rulesets like Clean Code), testing, debugging, source code management, deployment, etc. In addition, many other functional and non-functional properties, depending on the task at hand, might have to be checked, e.g. algorithmic suitability, to avoid performance problems with large data sets.

ChatGPT (and comparable AI tools) are quite proficient as a “Swiss army knife” in complex coding tasks - when used in an educated fashion. Their capabilities extend beyond single file programs, to ones featuring multiple classes and methods. A strong feature of ChatGPT is its ability to “scaffold” an application - the “hello world” creation, if you like - in an unfamiliar language or framework, like e.g. Spring Boot. This effectively reduces the frustrations of beginners to programming when learning new, complex technologies, since it can free them from spending a couple of days just getting the syntax right.

When problems arise, ChatGPT has also proven to be quite capable of identifying bugs and fixing them. This includes the ability to analyse stack traces - especially useful when working with a framework like Spring Boot<sup>9</sup>. Furthermore, the refactoring of existing code is also something that ChatGPT is quite proficient in. In one case, when asked to refactor an existing class to use some specific Clean Code rules in an SE2 assignment, the AI’s results were very close to perfection, only requiring some minor manual adjustments.

To ensure an *educated use* of AI tools, assignments in the “Unassisted Coding” phase need to train students in *AI Literacy*. This is defined by Long and Magerko [LM20] as “a set of competencies that enables individuals to critically evaluate AI technologies [...]”. The authors then describe 16 core competencies associated with AI Literacy. For the focus area of this paper (using AI in coding education), competency 5 (AI’s Strengths & Weaknesses) is especially important.

The definition by Long and Magerko sums up the key goal for actively embracing AI tools in SE curricula during this stage: “Identify problem types that AI excels at and problems that are more challenging for AI. Use this information to determine when it is appropriate to use AI and when to leverage human skills”. Regarding ChatGPT as a coding tool, this means that students can read and understand the generated code, and assess if it really fits to the task description given in the prompt. This competence fits to the “Unassisted Coding”

---

<sup>9</sup> Spring Boot is the de-facto industry standard for Java-based client-server applications (and should therefore be taught in SE curricula), but drives programming beginners mad with its infamous “stack traces from hell” [Nu12].

phase, where assignments need to include the *Evaluate* (5) Bloom level, to check code for the aforementioned aspects and quality attributes.

This would also include training the students in recognizing ChatGPT’s known weaknesses. As our case study (and other literature like [KG23]) shows, AI tools are (currently) error-prone when solving maths and logic assignments. A further weakness is the tendency of AI to add more complexity and to rewrite existing code in strange ways, when being asked to iterate on it. This tendency leads to harder to understand and more complex code, without any obvious advantages. Another weakness we observed is that working with AI and less well-known technologies yields worse results, with the AI having a greater tendency to “hallucinate” methods that don’t exist. Students need to actively use ChatGPT (and other AI tools) to gain a clear understanding of both their potential and their weaknesses.

This goal can be achieved by making the use of ChatGPT an *explicit part of the assignments*. There are manifold creative ways to include AI Literacy in the learning outcome of coding-focused modules. [Ge23a]<sup>10</sup>, [DB23], and [Ja23] make proposals in that direction (see the selection in table 2). Any interactions between ChatGPT and the student should also be documented and reflected upon, as a mandatory part of the solution.

Method	Source
Ask the students to let ChatGPT generate code, and then analyze and explain it	[Ja23]
Let students implement a solution using an alternative approach to ChatGPT’s solution	[Ja23]
Ask students for a different representation for ChatGPT’s solution, e.g. for a given data structure	[Ja23]
Let the students use ChatGPT for self assessment of their code, and ask them to reflect on ChatGPT’s improvement proposals	[DB23]
Let ChatGPT create coding tasks for students, and ask students to solve them	[DB23] [Ge23a]
Ask ChatGPT to create a faulty solution to a given problem, and then ask the students correct it	[Ge23a]
Let ChatGPT act as a teacher, explaining concepts and solutions	[Ge23a]
Switch roles between students and teacher: Give a reference solution to the students and instruct them to repeatedly prompt ChatGPT until it has generated a valid solution	[Ge23a]

Tab. 2: Innovative ways in which ChatGPT and similar tools can be used in SE education

As in stage 1, exams in coding-related courses in the “Unassisted Coding” phase need to be digital. They probably have to be two-phased: In phase one, access to ChatGPT needs to be open, so that students can work on assignments with explicit instructions to use AI tools, and document their interactions. In the second phase, with assignments focusing on other competencies, access must be blocked, since the scope of coding tasks is still

<sup>10</sup> This paper was written based on the results of the same teaching-research project as in this paper. The “DBS” module described there is the successor of “DB2” covered in this paper, in a subsequent examination regulation of the same study program.

relatively small, and ChatGPT would allow for an “uneducated” solution. A digital exam in a university’s computer lab would allow such a configuration switch with relative ease.

### **4.3 Stage 3 (Developing Complex Solutions): Use AI implicitly, without constraints**

The third and final stage allows the students unrestricted access to AI tools. During this stage, the scope of the tasks should be vast, to the point where using an “uneducated” approach would not be viable. This could be either done in large-scale practicals such as the ones in SE2 or DB2, or in large projects as found in the Computer Science Project (CSP), the Practical Project (PP) or Bachelor Thesis (BA). At this point, it would be up to the students to determine for which problems they would utilize the AI tools, as they should have learned in the previous stage how and where these tools perform best.

Exams are a great challenge in this phase. They need to be digital - as in the previous stages - and it would be natural to allow the students free access to AI tools during the exam. However, current exams are tailored towards a very limited time frame and assume no access to AI; these exams’ small scope would therefore make an “uneducated” approach possible, and thus would fail to assess the students’ true skills and knowledge. To still properly evaluate students when allowing them unrestricted use of AI tools, this scope needs to be enhanced. Unfortunately, this entails many changes, compared to the current exam design.

- With (far) larger assignments, manual corrections might not be viable anymore, due to the sheer lack of (human supervisor) resources. The correction therefore needs to be automatized to some extent - quite a challenge when testing functional and non-functional properties of written code.
- For fairness reasons, the students need to have free access to the same industry-standard AI tools they use outside of exams. Most AI tools are not free. Some (like GitHub Copilot) offer a free, or at least inexpensive, academic license. Others, like ChatGPT 4.0, do not even (at the time this paper is written) offer such a licensing model.
- An alternative would be to abort written exams altogether in this phase, and require project submissions instead. This, however, would greatly increase the workload on the supervisors, due to the large scope of the code they need to evaluate.

This situation requires further discussion and research. However, in the spirit of Constructive Alignment [Bi03] as a guiding didactical principle, the exam needs to reflect the learning outcome, which in turn should match the situation the students will face later in their professional context. So it seems we just need to solve this, one way or another.

## 5 Conclusion and Outlook

Based on our research findings, we recommend that SE curricula should be transformed, following the three stages proposed in figure 3. Modules teaching the fundamentals of programming should forbid AI tools entirely, while modules falling into stage 2 (unassisted coding) and stage 3 (developing complex solutions) need adjustment. Special attention should be given to the modules in stage 2, as these require the explicit usage of AI tools and thus the most work. Modules in stage 3 need to have a vast scope. Therefore, they will then be less affected by “uneducated” AI tool use. AI tools need to be available during the students’ examinations.

One limitation of our proposed conceptual framework is the simplified use of LoC to represent the scope dimension, as this does not fully represent the higher complexity of more advanced exercises. It also does not account for the shortcomings and strengths of AI tools as demonstrated with the SE2 practical in the summer semester of 2021 (see section 3.1). Further research should therefore explore other suitable attributes to represent scope. In addition, further validation of the model is required; systematic studies of coding education within other universities’ SE curricula, would help to verify its wider applicability and generalisability.

It should also be further explored how ChatGPT and other such tools could be included in creative ways within coding education - and the same applies to exam design in the age of AI tools. Finally, university teachers, researchers, and their professional organizations need to lobby for free educational licences of ChatGPT 4.0 and other industry standard AI tools. This is required to provide a level playing field for all students.

Ultimately, we feel that the glass is half full, rather than half empty, and that AI tools will be a great enrichment for Software Engineering education in general - if we manage to adapt our curricula accordingly.

## Bibliography

- [Ab23] Abdelfattah, Aly Maher; Ali, Nabila Ahmed; Elaziz, Mohamed Abd; Ammar, Hany H: Roadmap for Software Engineering Education using ChatGPT. In: 2023 International Conference on Artificial Intelligence Science and Applications in Industry and Society (CAISAIS). p. 1–6, September 2023.
- [AKB01] Anderson, Lorin W.; Kratwohl, David R.; Bloom, Benjamin Samuel: A Taxonomy for Learning, Teaching and Assessing: a Revision of Bloom’s Taxonomy. Longman, New York, 2001.
- [Bi03] Biggs, John: Aligning teaching for constructing learning. Higher Education Academy, pp. 1–4, 2003.
- [DB23] Daun, Marian; Brings, Jennifer: How ChatGPT Will Change Software Engineering Education. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. pp. 110–116, 2023.

- [DOH22] Dejean, Alexandre; Oedingen, Marc; Hammer, Maximilian: Artificial Intelligence-supported code completion. Research paper in module 'Coding Excellence' in Master Digital Sciences, 2022.
- [Ge23a] Geisel, Victoria; Schindler, Christian; Stein, Nils; Bente, Stefan: Lernräume unter Verwendung von generativen Sprachmodellen. paper submitted to SEUH 2024, 2023.
- [Ge23b] Geng, Chuqin; Yihan, Zhang; Pientka, Brigitte; Si, Xujie: Can ChatGPT Pass An Introductory Level Functional Language Programming Course? arXiv preprint arXiv:2305.02230, 2023.
- [HRL20] Hazzan, Orit; Ragonis, Noa; Lapidot, Tami: Guide to Teaching Computer Science: An Activity-Based Approach. Springer Nature, 2020.
- [Ja23] Jacques, Lorraine: Teaching CS-101 at the Dawn of ChatGPT. ACM Inroads, 14(2):40–46, may 2023.
- [Ji23] Jimenez, Carlos E; Yang, John; Wettig, Alexander; Yao, Shunyu; Pei, Kexin; Press, Ofir; Narasimhan, Karthik: SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv preprint arXiv:2310.06770, 2023.
- [KG23] Krüger, Tim; Gref, Michael: Performance of Large Language Models in a Computer Science Degree Program. arXiv preprint arXiv:2308.02432, 2023.
- [Ku23] Kuhail, Mohammad Amin; Mathew, Sujith Samuel; Khalil, Ashraf; Berenguere, Jose; Shah, Syed Jawad: "Will I Be Replaced?" Assessing Chatgpt's Effect on Software Development and Programmer Perceptions of Ai Tools. SSRN, 2023.
- [Li23] Li, Yihao; Xu, Jialong; Zhu, Yinghua; Liu, Huashuo; Liu, Pan: The Impact of ChatGPT on Software Engineering Education: A Quick Peek. In: 2023 10th International Conference on Dependable Systems and Their Applications (DSA). p. 595–596, August 2023.
- [LM20] Long, Duri; Magerko, Brian: What is AI Literacy? Competencies and Design Considerations. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. CHI '20, Association for Computing Machinery, New York, NY, USA, p. 1–16, 2020.
- [Ma23] Malinka, Kamil; Peresíni, Martin; Firc, Anton; Hujnak, Ondrej; Janus, Filip: On the educational impact of ChatGPT: Is Artificial Intelligence ready to obtain a university degree? In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. pp. 47–53, 2023.
- [Nu12] Nurkiewicz, Tomasz: , Filtering the Stack Trace From Hell, 2012.
- [SKM23] Sullivan, Miriam; Kelly, Andrew; McLaughlan, Paul: ChatGPT in higher education: Considerations for academic integrity and student learning. 2023.
- [We22] Welsh, Matt: The end of programming. Communications of the ACM, 66(1):34–35, 2022.
- [Zh23] Zhou, Aojun; Wang, Ke; Lu, Zimu; Shi, Weikang; Luo, Sichun; Qin, Zipeng; Lu, Shaoqing; Jia, Anya; Song, Linqi; Zhan, Mingjie; Li, Hongsheng: Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. arXiv e-prints, 2023.

# Lernräume unter Verwendung von generativen Sprachmodellen

## Ein Erfahrungsbericht zum kreativen Einsatz von Tools wie ChatGPT in der Software-Engineering-Hochschullehre

Victoria Geisel<sup>1</sup>, Christian Schindler<sup>2</sup>, Nils Stein<sup>3</sup>, Stefan Bente<sup>4</sup>

**Abstract:** University education may cultivate independent study by design of supportive environments for students. Recent proliferation of generative AI tools has had a significant impact on this. The extent to which successful scholarship is influenced by these tools is the subject of current research and the focus of this study. Tasks in the field of databases with different competence levels and didactic goals were solved by students with the help of GPT-3.5. Intellectual performance and success were then evaluated. The results show that integration of generative language models has the potential to improve academic attainment. Success proved highly dependent upon the nature of task and the strategy employed. The paper emphasises the need to integrate the effective use of AI tools into SE teaching, in order to use generative language models to their full potential in independent academic development.

**Keywords:** Künstliche Intelligenz; Hochschullehre; Bildungspraktiken; Lernprozesse; ChatGPT; Lernraum; generative Sprachmodelle; Software Engineering

## 1 Einleitung

Der kostenlose Zugang zu generativen Sprachmodellen hat die Wahrnehmung der Hochschul- und Berufswelt verändert. In einer Umfrage des Pew Research Center vom Juli 2023 in den USA [PG23] gaben 75% derjenigen Teilnehmer\*innen, die bereits von einem Chatbot wie beispielsweise ChatGPT gehört haben, an, dass ChatGPT in den nächsten 20 Jahren einen Einfluss auf den Beruf “Softwareentwickler\*in” haben wird. Die Studie zeigt ebenfalls auf, dass je höher der Bildungsgrad der Teilnehmenden ist, desto höher der relative Anteil an Personen, die daran glauben, dass ChatGPT einen Einfluss auf ihren eigenen Beruf haben wird. Als Mentoren und Ausbilder der Fachkräfte von morgen müssen Lehrende an Hochschulen und Universitäten die Aufklärung sowie Auseinandersetzung mit generativen Sprachmodellen in den Fokus rücken. Sind Sprachmodelle wie ChatGPT nur eine Gefahr

---

<sup>1</sup> Technische Hochschule Köln, Fakultät für Informatik und Ingenieurwissenschaften, victoria.geisel@th-koeln.de

<sup>2</sup> Technische Hochschule Köln, Fakultät für Informatik und Ingenieurwissenschaften, schindler.chris@icloud.com

<sup>3</sup> Technische Hochschule Köln, Fakultät für Informatik und Ingenieurwissenschaften, nils.stein@smail.th-koeln.de

<sup>4</sup> Technische Hochschule Köln, Fakultät für Informatik und Ingenieurwissenschaften, stefan.bente@th-koeln.de

für die bisherige Form der akademischen Lehre – oder ergeben sich durch einen kreativen Einsatz als Werkzeug auch neue Möglichkeiten des Kompetenzerwerbs?

Diese Frage wurde im Rahmen eines Lehrforschungsprojekts im Masterstudiengang *Digital Sciences* an der Fakultät für Informatik und Ingenieurwissenschaften der TH Köln untersucht [AL23]. Anhand des Moduls *Datenbanksysteme (DBS)* [HT23] im 4. Semester des Bachelorstudiengangs Informatik wurden verschiedene Lernräume mit individuellen didaktischen Zielen unter Einschluss von ChatGPT gestaltet. Diese Lernräume werden in den nachfolgenden Kapiteln dargestellt und in ihrer Wirksamkeit evaluiert. Dabei wird ausführlich auf die jeweiligen Vor- und Nachteile der Integration von ChatGPT eingegangen. Die Konzepte und Ergebnisse werden zwar anhand von Aufgabenstellungen aus dem relationalen Datenbankdesign diskutiert, lassen sich aber auch auf andere Bereiche des Software Engineerings übertragen.

Beim Forschungsdesign wurde der Umstand genutzt, dass einer der Autoren gleichzeitig wissenschaftlicher Mitarbeitender mit Verantwortung für die genannte Veranstaltung ist, und dass die verantwortlichen Professoren (Dr. Birgit Bertelsmeier und Dr. Johann Schaible) die Studie aktiv unterstützten. In diesem Rahmen wurden zwei Aufgabenblätter für diese Studie mit insgesamt sieben verschiedenen Aufgaben zusammengestellt, die unter Einschluss von ChatGPT zu bearbeiten waren. Semesterbegleitend haben dies im Sommersemester 2023 insgesamt 43 Studierende getan. Die Ergebnisse und Lösungswege (z.B. ChatGPT-Prompts) mussten dabei von den Studierenden dokumentiert werden. Diese boten, zusammen mit vier geführten Interviews und zahlreichen informell protokollierten Diskussionen in den Übungsgruppen, detailliertere Einblicke in die Vorgehensweisen und Prozesse bei der Bearbeitung dieser Lernräume. Die Ergebnisse sind in den nachfolgenden Kapiteln beschrieben. Zum Abschluss dieses Papers wird ein Fazit gezogen mit dem Ziel, eine Diskussionsgrundlage zu schaffen sowie zu weiterer Forschung anzuregen, wie der künftige Umgang mit KI-Tools an akademischen Lehrinrichtungen aussehen könnte.

## 2 Gestaltung von Lernräumen unter Einschluss von ChatGPT

Das Modul DBS befasst sich mit einfachen bis semi-komplexen Code-Strukturen im Datenbank-Bereich, größtenteils anhand der Programmiersprache PL/SQL. Kategorisiert in *Basisaufgaben*, *Vertiefende Aufgaben* und *Experimentelle Aufgaben* wurden die nachfolgend vorgestellten Lernräume wie folgt evaluiert:

- Die Autoren protokollierten die Ergebnisse der unsystematischen Selbstbeobachtungen, die sie beim eigenen Erstellen der Aufgabenblätter inklusive Musterlösungen gewinnen konnten, um vorab die Ergebnisqualität, den Lernerfolg und mögliche Stolpersteine identifizieren zu können. Die Ergebnisse der Beobachtungen mündeten in dem Forschungsdesign entsprechenden Anpassungen der Aufgabenstellungen.



- Die Studienteilnehmenden mussten nicht nur die eigene Lösung für die jeweiligen Aufgaben einreichen, sondern auch den gesamten in diesem Zusammenhang geführten Dialog mit ChatGPT. Teilweise wurde dieser noch von den Studierenden im Nachgang kommentiert.
- Im Verlauf der Studie gab es mehrere Online-Meetings mit den Studienteilnehmern, in denen die Aufgaben erläutert sowie über die Lösungen gesprochen wurde. Teilweise mündete dies in sehr detaillierten Diskussionen. Diese in Zoom durchgeführten Interviews folgten keinem definierten Leitfaden, um zum einen den persönlichen Austausch mit den Studierenden zu fördern, und zum anderen, um die Gesprächsthemen des noch unerforschten Bereichs nicht durch definierte Fragen einzuschränken. Die Interviews wurden informell protokolliert.
- Am Ende der Studie nahmen alle studentischen Teilnehmer anonym an einer Online-Umfrage teil. Dadurch wurde ihnen die Möglichkeit für ehrliches, vielleicht auch negatives oder kritisches Feedback gegeben. Die Studierenden wurden nach Aufwandschätzungen, Qualitätsbeurteilungen, Problemen und persönlichen Nutzungsverhalten gefragt.

Das Ziel der Studierenden im Zusammenhang mit diesem Experiment war die Generierung von lauffähigen Code-Lösungen mithilfe von GPT-3.5, die der Semantik der jeweiligen Aufgabenstellungen entsprachen. Im Detail bedeutet dies, dass die Vorgehensweise den Studierenden freigestellt war und durch iterative Anfragen an das generative Sprachmodell Aufgabenstellungen gelöst werden sollten. Für Aufgabenstellungen, die keine Generierung von Code erforderten, sollte durch iterative Anfragen eine korrekte Lösung für das in der Aufgabenstellung beschriebene Problem erzeugt werden. Die Motivation der Studierenden, gute Lösungen zu erarbeiten, wurde durch die Verteilung von Klausurbonuspunkten erhöht. Die Bewertung, ob eine Aufgabe korrekt gelöst ist, erfolgte durch die Autoren. Die im Folgenden verwendeten Prozentangaben zur Korrektheit einzelner Aufgaben beziehen sich auf diese Bewertungen.

## 2.1 Lernerfolg

Lernerfolg beschreibt den nachweisbaren Erwerb von Wissen und Fähigkeiten, in diesem Paper konkret beim selbstständigen Lernen im Rahmen der Lehre. Dabei können sowohl persönliche als auch vorgegebene Lernziele erreicht werden. Werden Lernziele nicht erreicht und Kompetenzen nicht erworben, liegt **nahezu kein Lernerfolg** vor. **Geringer Lernerfolg** bedeutet ansatzweises Erreichen von Lernzielen, wobei Kompetenzen nicht im ausreichenden Maß erworben werden. Bei **partielltem Lernerfolg** werden die Ziele nur in Teilen erreicht. Erfolgt ein **dependenzieller Lernerfolg** können Lernziele erreicht, sowie Kompetenzen unter Berücksichtigung von klar definierten Vorgehensweisen erworben werden. Bei einem **guten Lernerfolg** werden alle Lernziele erreicht und Kompetenzen vollumfänglich entsprechend des Learning Outcomes erworben.

## 2.2 Lernräume

Ein Lernraum bezeichnet die Umgebung, in der Lernaktivitäten stattfinden [St18]. Diese können physischer oder virtueller Natur sein. Insbesondere zum selbstständigen Lernen eignen sich virtuelle Lernräume, da keine Interaktion mit anderen Personen stattfindet. In einem digitalen Umfeld kann dieser Raum stark durch generative Sprachmodelle gestaltet und personalisiert werden.

Tab. 1: Übersichtstabelle Aufgaben

ID	Bloom Level	Aufgabe	Didaktisches Ziel	Ergebnisqualität	Lernerfolg
B1	3	Prozedur in PL/SQL programmieren	Abfrage und Anwendung von Grundlagen-Wissen in einem definierten Anwendungskontext	Nahezu perfekte Lösungen (98%)	Nahezu keiner
B2	3	Trigger in PL/SQL programmieren	Abfrage und Anwendung von Grundlagen-Wissen in einem definierten Anwendungskontext	Gute Lösungen (70%)	Gering
V1	4	Compound-Trigger programmieren	Erkennung von Zusammenhängen durch kritisches Auseinandersetzen mit dem Thema	Befriedigende Lösungen (60%)	Partiell
V2	4	Python-Skript für Oracle-Datenbank	Erkennung von Zusammenhängen durch kritisches Auseinandersetzen mit dem Thema	Gute bis sehr gute Lösungen (84%)	Dependenziell
E1	2	Fragen zu B-Bäumen beantworten	Nachhaltiges Lernen durch Erklären	Ausreichende Lösungen (51%)	Gering
E2.1	5	Fehlersuche in komplexer Prozedur, theoretische Fragen	Lernen durch Auffinden von Fehlern	Gute Lösungen (74%)	Gut
E2.2	2	Theoretische Fragen bekommen und selbst beantworten	Assistiertes Lernen durch selbstständiges Recherchieren in Literaturquellen	Gute Lösungen (74%)	Gering
E3	4	Trigger aus Grafik erstellen	Eigenständige Entwicklung sowie Interpretation einer nicht näher definierten Aufgabenstellung	Sehr gute Lösungen (86%)	Gut

Die Lernräume werden anhand von *Bloom's Revised Taxonomy* [AKB01] klassifiziert, mit deren Hilfe Lernziele in sechs Stufen (Wissen, Verstehen, Anwenden, Analysieren, Bewerten, Erschaffen) differenziert werden können. So kann sichergestellt werden, dass Lernende auf verschiedenen Ebenen des kognitiven Denkens gefordert werden. Tabelle 1 gibt einen Überblick über die gestellten Aufgaben, deren didaktisches Ziel sowie der Eignung der Aufgaben für das selbstständige Lernen von Studierenden gemessen an der Qualität der Ergebnisse und des Lernerfolgs.

### 3 Basisaufgaben (B)

Die in diesem Kapitel beschriebenen Aufgaben wenden Grundlagenwissen an bzw. fragen dieses ab. Die zu implementierenden Funktionalitäten waren einfach gehalten und sollen Aufgaben zum Einstieg in eine neue Programmiersprache repräsentieren. Beide gestellten Aufgaben dieses Lernraumes sind als Bloom-Level 3 einzuordnen. Die Studierenden sollen eine konkrete Technik in einem klar definiertem Umfeld anwenden. Das Lernziel ist dabei die Fähigkeit zu stärken, selbstständig theoretisches Wissen in einem Anwendungskontext praktisch umzusetzen.

Ziel der ersten Aufgabe (B1) war die Programmierung einer Prozedur ohne Rückgabewert. Die zweite Aufgabe (B2) fokussierte die Implementierung eines Triggers in PL/SQL mithilfe einer textuellen Beschreibung der Funktionalitäten und erforderte die Anwendung des Konzepts der Transition.

#### 3.1 Bewertung

98% der eingereichten Lösungen zur Prozeduraufgabe (B1) waren lauffähig und semantisch korrekt gelöst. Die aus den Gesprächsverläufen beobachtete Strategie der Studierenden war mehrheitlich das Kopieren und Einfügen der Aufgabenstellung an ChatGPT und es konnte in den meisten Fällen initial eine korrekte Lösung erfolgreich generiert werden.

70% der eingereichten Lösungen bei der Triggeraufgabe (B2) waren korrekt. Copy and paste der Aufgabenstellung führte seltener zu einer initial korrekten Lösung. ChatGPT hatte Probleme bei der Unterscheidung zwischen einer Transitionsvariable (bspw. OLD) und der Referenz auf eine Transitionsvariable (bspw. :OLD).<sup>5</sup> Es konnte ebenfalls beobachtet werden, dass der gewählte SQL-Dialekt sich im Verlauf des Gesprächs ohne separate Anweisung verändern kann (bspw. plötzlicher Wechsel zu MySQL als Antwort auf eine Anfrage).

Die vorgestellten Basisaufgaben unterstützen nach Evaluation der Studie nicht das selbstständige Lernen. Studierende empfanden den plötzlichen Wechsel des gewählten SQL-Dialekts als frustrierend. Wenn in Gesprächsverläufen ein Wechsel des Dialekts erkenntlich war, haben nur die wenigsten der betroffenen Studierenden dies korrekterweise erkannt und das generative Sprachmodell auf diesen Fehler aufmerksam gemacht. Mehrheitlich wurde dieser Fehler nicht erkannt und die Lösungen in einem falschen Dialekt eingereicht. Zusätzlich, durch den hohen Anteil an initial korrekt generierten Lösungen, konnte im Rahmen der Basisaufgaben kein hoher Lerneffekt beobachtet werden. Dieser wäre aber gegeben, wenn Studierende mehrheitlich den Wechsel des Dialekts verstanden sowie evaluiert hätten oder durch eigenständige Weiterentwicklung des Codes (bspw. Abfangen von möglichen Fehlerquellen) eine tiefgreifendere sowie qualitative Auseinandersetzung erfolgt wäre.

<sup>5</sup> Dieses Problem trat bei der verwendeten Version GPT-3.5 auf. Inwiefern es andere generative KI Modelle betrifft, etwa GPT-4.0, könnte in weiterführender Forschung untersucht werden.

## 4 Vertiefende Aufgaben (V)

Diese Aufgaben betrachteten erweiterte Datenbank-Konzepte. Zur Lösung dieser Aufgaben ist ein gewisses Vorwissen notwendig. Auf der Bloom-Skala sind beide Aufgaben als Level 4 einzuordnen. Die Studierenden analysierten ein konkretes Problem und lösten es mit einer passenden Technik. Sie mussten tiefer in das Material eintauchen und kritisch über die Anwendung und die Auswirkungen der Technik nachdenken, und lernten so, Strukturen und Zusammenhänge zu erkennen.

Eine Aufgabe befasste sich mit der Implementierung eines Compound-Triggers (V1), der die beiden in PL/SQL verfügbaren Arten (Statement- und Row-Trigger) verknüpft. Die zweite Aufgabe befasste sich mit der anwendungsseitigen Datenbankverknüpfung (V2) und verknüpfte zwei Programmiersprachen, indem ein Python-Skript entworfen werden musste, das sich mit einer Oracle-Datenbank verbindet und eine semantisch vorgegebene Query an die Datenbank abschickt.

### 4.1 Bewertung

Bei beiden Aufgaben konnte ChatGPT 3.5 im Rahmen der Studie keine initial korrekte Lösung generieren. 60% der eingereichten Lösungen zur Compound-Trigger Aufgabe (V1) waren korrekt und lauffähig. Bei dieser Aufgabe konnte mehrheitlich ein guter Lernprozess beobachtet werden, da das Kopieren und Einfügen der Aufgabenstellung nicht zum erwünschten Ziel führte und Studierende sich mit semantischem Fachwissen sowie mit dem generierten Code befassen mussten, um das Problem erkennen und beheben zu können. Es wurde auch beobachtet, dass ChatGPT bei optionalen Code-Strukturen leere Code-Blöcke erzeugen kann, die keine Funktionalitäten haben und somit nicht erforderlich wären, weshalb qualitativ minderwertiger Code erzeugt wird.

Die Aufgabe zur Anbindung einer Datenbank an eine Anwendung (V2) konnte zu 84% korrekt gelöst werden. Die Verknüpfung von zwei Programmiersprachen in einem Skript scheint für ChatGPT kein Problem darzustellen, die vorgeschlagenen Lösungen hatten aber kleine Mängel, die korrigiert werden mussten. Das Kopieren und Einfügen von Fehlermeldungen, die beim Testen des Codes ausgegeben werden, führte bei ChatGPT zu einer korrekten Lösung, was auch die mehrheitlich von den Studierenden verwendete Strategie darstellte. Dies könnte dazu führen, dass der Lernerfolg auf Seiten der Studierenden ausbleibt.

Die vorgestellten vertiefenden Aufgaben unterstützen nach Evaluation der Studie das selbstständige Lernen bedingt. Der Lernerfolg bei der Programmierung eines Compound-Triggers (V1) zeigte sich deutlich in den eingereichten Chatverläufen der Teilnehmenden. Es konnte mehrfach beobachtet werden, dass die Strategie, die Aufgabenstellung oder die Oracle-Fehlermeldungen zu übergeben, nicht mehr funktioniert hat. In diesem Fall mussten die Fehler eigenhändig in dem generierten Codeausschnitt gesucht werden, und somit war

eine tiefgreifendere Auseinandersetzung mit dem Code erforderlich. Es konnte ebenfalls beobachtet werden, dass nach Scheitern dieser Strategie vermehrt Verständnisfragen zur Semantik des Codes gestellt wurden, die zum Lernerfolg beitrugen. Wie bereits bei der Trigger-Aufgabe (B1) konnte in der Compound-Trigger Aufgabe (V1) beobachtet werden, dass ChatGPT Schwierigkeiten mit Transitionen hatte, zusätzlich verstärkt durch Verwendung von Transitionen in Codeblöcken, die keine Transitionen erlaubten.

In der zweiten Aufgabe (V2) hing der Lernerfolg stark von der Strategie der Studierenden ab, da das Kopieren und Einfügen der Fehlermeldungen von Oracle zu einem Erfolg führen konnte. Bei Eintragung benutzerspezifischer Daten (sowie Serverdaten) zeigten Studierende Kreativität und modifizierten durch Folgeanfragen an ChatGPT die generierte Lösung so, dass das Auslesen der Daten aus einer Datei erfolgte oder über einen Input-Wert bei der Ausführung des Skripts eingetragen werden musste. Vereinzelt gab es auch Studierende, die persönliche Login-Daten der zur Verfügung gestellten Datenbankinstanz an ChatGPT zur Generierung einer Lösung übergaben.

## 5 Experimentelle Aufgaben (E)

Experimentelle Aufgaben hatten den Zweck, kreative Aufgabenstellungen und das Potenzial von generativen Sprachmodellen zu untersuchen. Zwei Aufgaben fokussierten ein Rollenspiel zwischen Sprachmodell und Studierenden, bei dem die Rollen der Studierenden und Lehrenden eingenommen wurden. Die beiden Aufgaben verfolgten das didaktische Ziel, einen Dialog zwischen Lernenden und Lehrenden zu schaffen, wie er außerhalb des digitalen Lernumfelds zu finden ist. Durch das Einnehmen der Rolle als Lehrende durch die Studierenden sollte der bewährte Ansatz erprobt werden, bei dem sich bei Lernenden durch das Erklären von Inhalten an Dritte ein nachhaltiger Lernerfolg einstellt [Ma02]. In einem entgegengesetzten Ansatz sollte ChatGPT den Lernenden eine falsche Antwort liefern, sodass sich aus dem Auffinden von Fehlern ein Lernerfolg einstellen konnte. Des Weiteren sollte ChatGPT den Lernenden Fragen zu theoretischen Grundlagen stellen. Nach dem Prinzip „Assistiertes Lernen“ sollte ChatGPT den Lernenden einen thematisch relevanten Quellverweis liefern, der diese bei der Beantwortung der Frage unterstützt. Die Komplexität der Lernziele befindet sich dabei auf unterschiedlichen Stufen (einfachem Verstehen (2), Analysieren (4) und Beurteilen (5)). In der letzten Aufgabe sollten Studierende auf Basis eines visuellen Inputs selbstständig kreativ agieren und eine Aufgabenstellung samt Lösung entwickeln. Die Aufgabe wurde auf Grundlage der Inquiry-Based Learning Methode entwickelt [SAC23]. Dabei soll durch Erfahrungswerte und tiefgründiges Hinterfragen der verfügbaren Informationen eigenständig Wissen ermittelt werden.

Die erste Aufgabe verfolgte den Ansatz, die Rollen von Studierenden und ChatGPT zu vertauschen (E1). Es wurde die zum Zeitpunkt der Studie vorhandene mathematische Schwäche [Zh23] von ChatGPT verwendet, damit Studierende die Perspektive eines Lehrenden einnehmen und versuchen konnten, anhand von Erklärungen mathematischer Formeln das Sprachmodell zu einer korrekten Lösung zu führen. Die Aufgabenstellung

setzte sich mit binären Bäumen, bspw. mit der Fragestellung nach der minimalen Höhe eines binären Baumes unter bestimmten Bedingungen, auseinander. Studierenden war Zugriff auf eine Musterlösung gewährt. Diese sollte nicht an ChatGPT übergeben werden, sondern durch iterative Anfragen auf die vorgeschlagene Lösung von ChatGPT Stellung zu nehmen und durch Hinweise zu unterstützen.

In der zweiten Aufgabe sollten Studierende einen vorgegebenen Prompt an ChatGPT versenden, der eine entsprechende zufällige Aufgabe generiert. Diese sollten die Studierenden anschließend lösen (E2). In der ersten Teilaufgabe (E2.1) sollte eine komplexe Prozedur mit bewusst eingebauten Fehlern generiert werden und Studierende hatten die Aufgabe diese Fehler zu finden. In der zweiten Teilaufgabe (E2.2) sollten fünf theoretische Fragen zu PL/SQL, mit einem Hinweislink versehen, von ChatGPT gestellt und von den Studierenden beantwortet werden. Die gestellten Fragen waren mehrheitlich nicht spezifisch, sondern allgemein gehalten und beschäftigten sich z.B. mit dem Unterschied zwischen einer Prozedur und einer Funktion in PL/SQL, oder wofür ein Cursor benötigt wird.

In der dritten Aufgabe sollten Studierende selbst eine Aufgabe konstruieren und lösen, ohne kopierfähige Informationen an ChatGPT weitergeben zu können (E3). Hierfür wurden den Studierenden visuelle Informationen in Form eines Bildes zur Verfügung gestellt, damit sie einen Trigger mithilfe der verfügbaren Informationen implementieren.<sup>6</sup>

## 5.1 Bewertung

51% der eingereichten Lösungen der B-Baum-Aufgabe (E1) wurden als korrekt deklariert. Der niedrige Anteil daran lässt sich damit erklären, dass die Aufgabe als sehr frustrierend empfunden wurde, da das generative Sprachmodell bspw. die Vorschläge zur Berechnung der korrekten Höhe teilweise ignorierte oder nicht umsetzte. Diese Aufgabe erforderte auch eine andere Herangehensweise von den Studierenden, da das Tool die richtige Lösung nicht kannte und andere Hilfsmittel als Werkzeuge mit eingebunden werden mussten. Nach Gesprächen mit Studierenden konnte festgestellt werden, dass die intensiven Trial-and-Error Versuche zu einer tiefgreifenden Auseinandersetzung mit der Thematik führten, aber der Nutzen den Frust nicht überwog.

74% der Studierenden reichten eine korrekte Lösung bei der zweiten Aufgabe (E2) ein. Nach Evaluation der ersten Teilaufgabe (E2.1) konnte festgestellt werden, dass die Qualität der von ChatGPT generierten Aufgaben sehr stark schwankte. Es wurden sowohl Aufgaben generiert, die mehrere Fehler angemessener Schwierigkeit beinhalteten, als auch Aufgaben,

---

<sup>6</sup> Die von den Autoren initial überlegte Aufgabenstellung lautete wie folgt: Eine Kundin tätigt eine Bestellung bei einem Händler und wartet auf ihr Paket. In Abhängigkeit von der benötigten Lieferzeit in Tagen ergeben sich unterschiedliche Szenarien. Wenn die Zustellung des Pakets an die Kundin in unter zehn Tagen erfolgt, gilt das als erfolgreiche Zustellung, die keine weiteren Handlungen auslöst. Wenn das Paket länger als zehn, aber weniger als 20 Tage benötigt, bekommt die Kundin einen Gutschein/Rückerstattung im Wert von 20 Euro. Sollte die Lieferung länger als 20 Tage in Anspruch nehmen, wird der Kundin ein Gutschein/Rückerstattung im Wert von 50 Euro gewährt.

die keine, unlogische oder nur einen Fehler beinhalteten, der sich lediglich auf Tippfehler bezog. Es kam ebenfalls vermehrt vor, dass Fehler bereits beim Stellen der Aufgabe an die Studierenden aufgelöst und erklärt wurden. Bei der zweiten Teilaufgabe (E2.2) kam es vermehrt vor, dass die Hinweislinks fehlerhaft waren. Verfügbare Links entstammten offiziellen Quelle (bspw. Handbuch von Oracle). Auch bei dieser Teilaufgabe erläutert ChatGPT gelegentlich die Lösung bereits in der Fragestellung. Die gestellten Fragen waren mehrheitlich nicht spezifisch, sondern allgemein gehalten.

86% der eingereichten Lösungen zu Aufgabe 3 (E3) waren korrekt. 30% der Teilnehmer missverstanden die initiale Aufgabenstellung und konstruierten teilweise Trigger, die keine sinnvolle Semantik aufwiesen. Der hohe Anteil an korrekten Lösungen lässt sich dadurch erklären, dass eine sinnvolle Semantik kein Teil der Aufgabenstellung war, sondern, wie auch bei den übrigen Aufgaben, ein lauffähiger Code als Endergebnis gefordert wurde. Studierende zeigten bei dieser Aufgabenstellung mehrheitlich Kreativität bei der Gestaltung der Aufgabe. Studierende tendierten vermehrt dazu, Verbesserungen an dem generierten Code vorzunehmen. Vermutlich war das Fehlen einer konkreten Aufgabenstellung sowie die gegebene Freiheit bei der Generierung einer Lösung ein Motivator dafür, sich mit dem eigenen Code intensiver auseinanderzusetzen. In der Aufgabenstellung war zudem absichtlich ein Logikfehler eingebaut, denn es gab keine Fälle für genau zehn bzw. genau 20 Tage. Es sollte überprüft werden, ob diese entgegen der Aufgabenstellung von ChatGPT, oder den Studierenden berücksichtigt werden – beides war nicht der Fall.

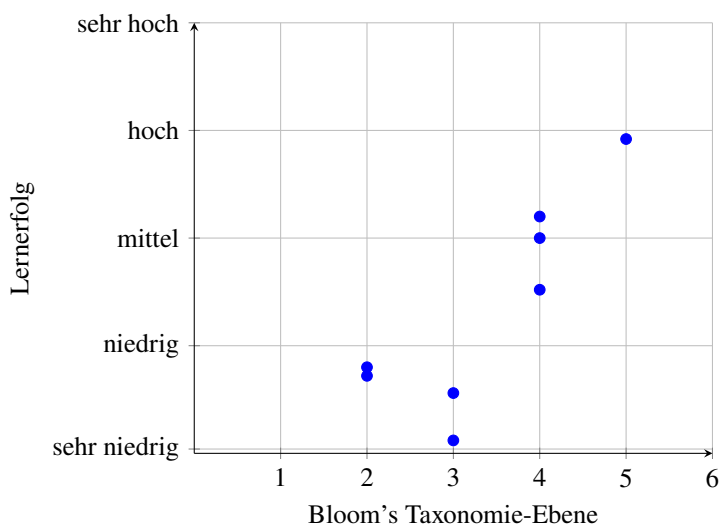
Die vorgestellten experimentellen Aufgaben unterstützen nach Evaluation der Studie das selbstständige Lernen bedingt. Das Ausnutzen der mathematischen Schwäche von ChatGPT gestaltete sich als schwierig, da die Studierende sich zwar einerseits tiefgreifend mit dem Thema mithilfe anderer Quellen auseinandersetzen mussten, um die Rolle eines Lehrenden einzunehmen. Andererseits aber löste dies bei den Studierenden eine hohe Frustration aus, da ChatGPT trotz richtiger Erklärungen und Vorschlägen zur Berechnung seitens der Studierenden Schwierigkeiten hatte, eine korrekte Lösung zu generieren. Diese Frustration scheint den Lernerfolg zu überwiegen, weshalb die Aufgabenstellung in dieser Form nicht zu Zwecken des selbstständigen Lernens empfohlen werden kann. Zudem ist es sehr wahrscheinlich, dass die mathematischen Fähigkeiten von ChatGPT sich mit der Einführung von Code Interpreter [Zh23] verbessern werden.

Als Verbesserung und Idee für weiterführende Studien könnte ChatGPT dazu aufgefordert werden, sich wie ein Studierender zu verhalten und Rückfragen zu einem bestimmten Sachverhalt zu stellen, der dann von den Studierenden erläutert werden muss. Die Bearbeitung von zufällig generiertem Content, wie in der zweiten Aufgabenstellung, wird als eine gute Möglichkeit für das selbstständige Lernen angesehen. Es ist zwar möglich, dass nicht gelehrt Inhalte in diesem Kontext abgefragt werden können, was jedoch durch eine Einschränkung in einer detaillierten Anfrage an das generative Sprachmodell umgangen werden kann. Obwohl beobachtet werden konnte, dass Lösungen teils direkt bei der Fragestellung übermittelt wurden, wird der Einfluss auf den Lernerfolg eher als gering betrachtet. Einerseits können die Informationen nach wie vor entnommen und verinnerlicht werden. Andererseits sollten die

Lösungen von ChatGPT nach wie vor evaluiert werden, da initial nicht von einer korrekten Antwort ausgegangen werden konnte. Ein Lernerfolg war erkennbar, wenn Studierende die Plausibilität der Lösungen eigenständig beurteilen mussten.

Die dritte Aufgabe, die sich mit der Entnahme visueller Informationen beschäftigte, kann auch als eine gute Möglichkeit für das selbstständige Lernen gelten. Studierende mussten die gegebenen Informationen verschriftlichen und eine Anfrage an das generative Sprachmodell stellen. Dabei mussten sie die Semantik der Aufgabe verstehen, um plausible Lösungen generieren zu lassen. Auch die Kreativität sowie selbstständig durchgeführte Verbesserungen an der Aufgabe sprechen für eine positive Einstellung der Studierenden bei der Bearbeitung. Allerdings sollte beachtet werden, dass die Interpretation der Aufgabenstellung von der initial intendierten Aufgabe stark abweichen kann, und es nicht die *eine* korrekte Lösung gibt. Zudem kann man davon ausgehen, dass das Auslesen eines Bildes mit ChatGPT zukünftig möglich sein wird, was den Lernerfolg durch die Möglichkeit des Kopierens und Einfügens der Aufgabenstellung in den Prompt stark einschränken würde.

## 6 Abhängigkeit Taxonomie – Lernerfolg



Die Ergebnisse der Fallstudie zeigen, dass höhere Ebenen der Bloom'schen Taxonomie tendenziell mit einem höheren Lernerfolg verbunden sind.<sup>7</sup> Dies deutet darauf hin, dass

<sup>7</sup> Bei einem unterdurchschnittlichen Lernerfolg überwiegen die Nachteile des Einsatzes von generativer KI, da eigene Kompetenzen durch gut generierte Inhalte nicht ausreichend abgefragt werden. Im Gegensatz dazu konnten bei der Einstufung in der oberen Diagrammhälfte Vorteile festgestellt werden, da die Integration des eigenen Fachwissens eine entscheidende Rolle spielt. Die dargestellten Werte sind eine Einschätzung der Autoren, basierend auf den im Laufe des Projekts gesammelten Daten.



Aufgaben, die komplexere kognitive Fähigkeiten erfordern und mit generativer KI gelöst werden, zu einem größeren Lernerfolg führen können. Dennoch ist der Lernerfolg auch stark abhängig von verschiedenen Faktoren wie der Art der Aufgabe, der Lernumgebung und den individuellen Lernstrategien der Studierenden, wie die gewisse Variabilität zeigt. Andere Faktoren, wie die spezifischen Lernziele der Aufgabe, die Motivation und das Engagement der Studierenden, sowie die Unterstützung und Ressourcen, die den Studierenden zur Verfügung stehen, spielen ebenfalls eine wichtige Rolle. Auch die Größe der betrachteten Aufgabenstellung wäre noch interessant, was in einem weiteren Paper aus diesem Lehrforschungsprojekt [BRW23] vertieft wird. Weitere Forschung ist hier notwendig, um einen möglichen Zusammenhang nachzuweisen. Weiteres Analysepotential liegt zum einen in einer möglichen Korrelation der Aufgabenart und dem Lernerfolg und zum anderen, inwiefern die Strategie der Studierenden den Lernerfolg beeinflusst.

## 7 Fazit

Die Studie konnte in einem Modul an einer deutschen Hochschule am Beispiel von ChatGPT zeigen, dass die Integration von generativen Sprachmodellen in den akademischen Lehrkontext das Potenzial hat, den Lernerfolg von Studierenden zu verbessern. Der Erfolg hängt jedoch stark von der Strategie und Motivation der Studierenden ab. Wenn Studierende sich dafür intrinsisch motiviert entscheiden, eine vorgeschlagene Lösung des generativen Sprachmodells zu evaluieren und nicht ohne Weiteres zu übernehmen, können Lernerfolge auch bei einfachen Fragestellungen gemessen werden.

Die Basisaufgaben zeigten, dass die Studierenden in der Lage waren, grundlegende Konzepte erfolgreich mit ChatGPT anzuwenden. Diese Aufgaben förderten jedoch nicht unbedingt das selbstständige Lernen, da ein einfaches Copy and paste möglich war und ein intensives Befassen mit der Aufgabenstellung bzw. der Thematik nicht nötig war.

Die vertiefenden Aufgaben erforderten trotz ChatGPT-Nutzung ein tieferes Verständnis der Thematik und förderten das selbstständige Lernen. Allerdings hing der Lernerfolg stark von der Strategie der Studierenden ab. Um einen wirklichen Mehrwert zu schaffen, ist es dafür in Zukunft sinnvoll, den Umgang mit LLMs sowie Prompt Engineering zu lehren.

Die experimentellen Aufgaben zeigten ebenfalls das Potenzial von generativen Sprachmodellen zur Förderung des selbstständigen Lernens auf. Auch wenn etwa das Rollenspiel zwischen ChatGPT und Studierenden zu Frustrationen führte, konnte durch das Generieren von zufälligem Inhalt und das Verarbeiten visueller Informationen ein Bereich gezeigt werden, in dem das selbstständige Lernen unterstützt werden kann.

Insgesamt zeigt die Studie an verschiedenen Beispielen, dass die Integration von generativen Sprachmodellen im akademischen Lehrkontext das Potenzial hat, den Lernerfolg zu verbessern. Allerdings ist weitere Forschung und Diskussion nötig, da es sich um einen auf den Hochschul- und Fachbereich beschränkten Erfahrungsbericht handelt. Weitere

Forschung ist notwendig, um praktische Leitlinien mit detaillierteren Empfehlungen zum Einsatz von KI-Werkzeugen in der Lehrpraxis zu erarbeiten, einschließlich möglicher Fallstricke und bewährter Verfahren.

Lehrende sollten Lernende dazu ermutigen, kritisch zu denken und selbstständig Lösungen zu finden, anstatt sich auf eine generierte Lösung zu verlassen. Erreicht werden könnte dies durch Aufgabenstellungen, die ein höheres Bloom-Level haben, ein tieferes Verständnis erfordern und durch das Lehren von Strategien, die Studierende dazu ermutigen, vorgeschlagene Lösungen kritisch zu hinterfragen, im Kontext einzuordnen und zu verbessern.

In Anbetracht der rasanten Fortschritte in der Künstlichen Intelligenz ist es unerlässlich, dass sich Lehrende nicht prinzipiell gegen diese Entwicklung stellen, sondern versuchen, sie zu beherrschen. Die Nutzung von Tools wie ChatGPT wird immer verbreiteter. Ein Verbot ist weder praktikabel noch wünschenswert. Die Lehre sollte sich darauf konzentrieren, Studierende darauf vorzubereiten, entsprechende Tools verantwortungsvoll zur Verbesserung ihres Lernens und zur Erweiterung ihrer Fähigkeiten nutzen zu können.

## Literatur

- [AKB01] Anderson, L. W.; Kratwohl, D. R.; Bloom, B. S.: A Taxonomy for Learning, Teaching and Assessing: a Revision of Bloom's Taxonomy. Longman, New York, 2001.
- [AL23] ArchiLab, 2023, URL: [https://www.archi-lab.io/projects/ss23/gp\\_chatgpt\\_ss23.html](https://www.archi-lab.io/projects/ss23/gp_chatgpt_ss23.html), Stand: 17. 10. 2023.
- [BRW23] Bente, S.; Randall, N.; Wäckerle, D.: A Conceptual Framework to Transform Coding Education in Times of Generative AI, SEUH 2024, 2023.
- [HT23] Köln, H. T., 2023, URL: <https://hops.gm.th-koeln.de/hops/modules/modulelisting/module.php?mkz=1577>, Stand: 03. 11. 2023.
- [Ma02] Martin, J.-P.: Lernen durch Lehren (LdL). Die Schulleitung - Zeitschrift für pädagogische Führung und Fortbildung in Bayern/, 2002.
- [PG23] PARK, E.; GELLES-WATNICK, R.: Most Americans haven't used ChatGPT; few think it will have a major impact on their job. Pew Research Center/, 2023.
- [SAC23] College, S. A., 2023, URL: <https://sac.edu/AcademicAffairs/TracDat/Pages/Inquiry-Based-Learning-.aspx>, Stand: 12. 11. 2023.
- [St18] Stang, R.; Bernhard, C.; Kraus, K.; Schreiber-Barsch, S.: Lernräume in der Erwachsenenbildung. In (Tippelt, R.; von Hippel, A., Hrsg.): Handbuch Erwachsenenbildung/Weiterbildung. Springer Fachmedien Wiesbaden, Wiesbaden, S. 643–658, 2018.
- [Zh23] Zhou, A.; Wang, K.; Lu, Z.; Shi, W.; Luo, S.; Qin, Z.; Lu, S.; Jia, A.; Song, L.; Zhan, M.; Li, H.: Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. arXiv e-prints/, 2023.

# Autor:innenverzeichnis

## **B**

Becker, Steffen, 53  
Bente, Stefan, 93, 105  
Bockisch, Christoph, 65  
Böttcher, Axel, 13

## **D**

Dick, Steffen, 65  
Dreyer, Teresa, 65

## **G**

Geisel, Victoria, 105  
Gross, Thomas R., 85

## **H**

Hoorn, André van, 39

## **K**

Kieslinger, Julian, 53  
Körndle, Hermann, 25

## **L**

Luedtke, Bernhard M., 77  
Lüttgen, Gerald, 77

## **M**

Meißner, Niklas, 53

## **R**

Rachow, Paula, 39  
Rahe, Christian, 39  
Randall, Natasha, 93  
Ringel, Robert, 25

## **S**

Schindler, Christian, 105  
Speth, Sandro, 53  
Stein, Nils, 105

## **T**

Turner, Veronika, 13

## **W**

Wäckerle, Dennis, 93

## **Y**

Yip, Eugene, 77

## *GI-Edition Lecture Notes in Informatics*

- P-313 Ludger Humbert (Hrsg.)  
Informatik – Bildung von Lehrkräften in  
allen Phasen  
19. GI-Fachtagung Informatik und Schule  
8.–10. September 2021 Wuppertal
- P-314 Gesellschaft für Informatik e.V. (GI)  
(Hrsg.)  
INFORMATIK 2021 Computer Science &  
Sustainability  
27. September– 01. Oktober 2021, Berlin
- P-315 Arslan Brömme, Christoph Busch,  
Naser Damer, Antitza Dantcheva,  
Marta Gomez-Barrero, Kiran Raja,  
Christian Rathgeb, Ana F. Sequeira,  
Andreas Uhl (Eds.)  
BIOSIG 2021  
Proceedings of the 20th International  
Conference of the Biometrics  
Special Interest Group  
15.–17. September 2021  
International Digital Conference
- P-316 Andrea Kienle, Andreas Harrer,  
Jörg M. Haake, Andreas Lingnau (Hrsg.)  
DELFI 2021  
Die 19. Fachtagung Bildungstechnologien  
der Gesellschaft für Informatik e.V.  
13.–15. September 2021  
Online 8.–10. September 2021
- P-317 M. Gandorfer, C. Hoffmann, N. El Benni,  
M. Cockburn, T. Anken, H. Floto (Hrsg.)  
Informatik in der Land-, Forst- und  
Ernährungswirtschaft  
Fokus: Künstliche Intelligenz in der Agrar-  
und Ernährungswirtschaft  
Referate der 42. GIL-Jahrestagung  
21.–22. Februar 2022 Agroscope, Tänikon,  
Ettenhausen, Schweiz
- P-318 Andreas Helferich, Robert Henzel,  
Georg Herzwurm, Martin Mikusz (Hrsg.)  
FACHTAGUNG SOFTWARE  
MANAGEMENT 2021  
Fachtagung des GI-Fachausschusses  
Management der Anwendungsentwicklung  
und -wartung im Fachbereich Wirtschafts-  
informatik (WI-MAW), Stuttgart, 2021
- P-319 Zeynep Tuncer, Rüdiger Breitschwerdt,  
Helge Nuhn, Michael Fuchs, Vera Meister,  
Martin Wolf, Doris Weßels, Birte Malzahn  
(Hrsg.)  
3. Wissenschaftsforum:  
Digitale Transformation (WiFo21)  
5. November 2021 Darmstadt, Germany
- P-320 Lars Grunske, Janet Siegmund,  
Andreas Vogelsang (Hrsg.)  
Software Engineering 2022  
21.–25. Februar 2022, Berlin/Virtuell
- P-321 Veronika Thurner, Barne Kleinen, Juliane  
Siegeris, Debora Weber-Wulff (Hrsg.)  
Software Engineering im Unterricht der  
Hochschulen SEUH 2022  
24.–25. Februar 2022, Berlin
- P-322 Peter A. Henning, Michael Striewe,  
Matthias Wölfel (Hrsg.)  
DELFI 2022 Die 20. Fachtagung  
Bildungstechnologien der Gesellschaft für  
Informatik e.V.  
12.–14. September 2022, Karlsruhe
- P-323 Christian Wressnegger, Delphine  
Reinhardt, Thomas Barber, Bernhard C.  
Witt, Daniel Arp, Zoltan Mann (Hrsg.)  
Sicherheit 2022  
Sicherheit, Schutz und Zuverlässigkeit  
Beiträge der 11. Jahrestagung des  
Fachbereichs Sicherheit der Gesellschaft  
für Informatik e.V. (GI)  
5.–8. April 2022, Karlsruhe
- P-324 Matthias Riebisch,  
Marina Tropmann-Frick (Hrsg.)  
Modellierung 2022  
Fachtagung vom 27. Juni - 01. July 2022,  
Hamburg
- P-325 Heiko Roßnagel,  
Christian H. Schunck,  
Sebastian Mödersheim (Hrsg.)  
Open Identity Summit 2022  
Fachtagung vom 07. - 08. July 2022,  
Copenhagen
- P-326 Daniel Demmler, Daniel Krupka, Hannes  
Federrath (Hrsg.)  
INFORMATIK 2022  
26.–30. September 2022  
Hamburg
- P-327 Masud Fazal-Baqaie, Oliver Linssen,  
Alexander Volland, Enes Yigitbas,  
Martin Engstler, Martin Bertram,  
Axel Kalenborn (Hrsg.)  
Projektmanagement und  
Vorgehensmodelle 2022  
Trier 2022
- P-328 Volker Wohlgemuth, Stefan Naumann,  
Hans-Knud Arndt, Grit Behrens,  
Maximilian Höb (Editors)  
Environmental Informatics 2022  
26.–28. September 2022,  
Hamburg, Germany

- P-329 Arslan Brömme, Naser Damer, Marta Gomez-Barrero, Kiran Raja, Christian Rathgeb, Ana F. Sequeira, Massimiliano Todisco, Andreas Uhl (Eds.) BIOSIG 2022  
14. - 16. September 2022, International Conference
- P-330 Informatik in der Land-, Forst- und Ernährungswirtschaft  
Fokus: Resiliente Agri-Food-Systeme  
Referate der 43. GIL-Jahrestagung  
13.–14. Februar 2023 Osnabrück
- P-331 Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, Gottfried Vossen (Hrsg.)  
Datenbanksysteme für Business, Technologie und Web (BTW 2023)  
06.–10. März 2023, Dresden
- P-332 Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.)  
Software Engineering 2023  
20.–24. Februar 2023, Paderborn
- P-333 Steffen Becker & Christian Gerth (Hrsg.)  
SEUH 2023  
23.–24. Februar 2023, Paderborn
- P-334 Andreas Helferich, Dimitri Petrik, Gero Strobel, Katharina Peine (Eds.)  
1<sup>st</sup> International Conference on Software Product Management  
Organized by „GI Fachgruppe Software Produktmanagement im Fachbereich Wirtschaftsinformatik (WI PrdM)“  
Frankfurt, 2023
- P-335 Heiko Roßnagel, Christian H. Schunck, Jochen Günther (Hrsg.)  
Open Identity Summit 2023  
15.–16. June 2023, Heilbronn
- P-336 Lutz Hellmig, Martin Hennecke (Hrsg.)  
Informatikunterricht zwischen Aktualität und Zeitlosigkeit  
20.-22. September 2023, Würzburg
- P-337 Maike Klein, Daniel Krupka, Cornelia Winter, Volker Wohlgemuth (Hrsg.)  
INFORMATIK 2023  
Designing Futures: Zukünfte gestalten  
26. – 29. September 2023, Berlin
- P-338 René Röpke und Ulrik Schroeder (Hrsg.)  
21. Fachtagung  
Bildungstechnologien (DELFI)  
11.-13. September 2023, Aachen
- P-339 Naser Damer, Marta Gomez-Barrero, Kiran Raja, Christian Rathgeb, Ana F. Sequeira, Massimiliano Todisco, Andreas Uhl (Eds.)  
BIOSIG 2023  
20.-22. September 2023, Darmstadt
- P-340 Axel Kalenborn, Masud Fazal-Baqaie, Oliver Linssen, Alexander Volland, Enes Yigitbas, Martin Engstler, Martin Bertram (Hrsg.)  
Projektmanagement und Vorgehensmodelle 2023  
16. und 17. November 2023, Hagen
- P-341 Gunnar Auth und Tim Pidun (Hrsg.)  
6. Fachtagung Rechts- und Verwaltungsinformatik (RVI 2023)  
26.–27. Oktober 2023, Dresden
- P-342 Volker Wohlgemuth, Dieter Kranzlmüller, Maximilian Hüb (Eds.)  
EnviroInfo 2023  
11.–13. October 2023, Garching, Germany
- P-343 Rick Rabiser, Manuel Wimmer, Iris Groher, Andreas Wortmann, Bianca Wiesmayr (Eds.)  
Software Engineering 2024  
26. Februar – 1. März 2024, Linz
- P-344 C. Hoffmann, A. Steinm E. Gallmann, J. Dörr, C. Krupitzer, H. Floto (Hrsg.)  
Informatik in der Land-, Forst- und Ernährungswirtschaft  
27.–28. Februar 2024, Stuttgart-Hohenheim
- P-346 Axel Schmolitzky, Stefan Klikovits (Hrsg.)  
SEUH 2024  
29. Februar – 01. März 2024  
Linz, Österreich

All volumes of Lecture Notes in Informatics can be found at  
<https://dl.gi.de/handle/20.500.12116/21>.

The titles can be purchased at:

**Köllen Druck + Verlag GmbH**

Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn

Fax: +49 (0)228/9898222

E-Mail: [druckverlag@koellen.de](mailto:druckverlag@koellen.de)

Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in co-operation with GI and to publish the annual GI Award dissertation.

Broken down into

- seminars
- proceedings
- dissertations
- thematics

current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: <http://www.gi.de/service/publikationen/lni/>

ISSN 1617-5468

ISBN 978-3-88579-740-1

SEUH 2024 is the twentieth event in a conference series designed for educators from universities and universities of applied sciences to convene and exchange ideas on software engineering education. The conference provides a forum for discussing experiences and strategies, with a primary focus on enhancing teaching quality in the field. This volume contains contributions from the refereed main program, focusing topics such as heterogeneous cohorts, artificial intelligence, teaching programming in times of generative AI, and others.