Konzeption und prototypische Entwicklung eines computergestützten Systems zur Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering

Inaugural-Dissertation zur Erlangung der Doktorwürde der Fakultät für Sprach-, Literatur- und Kulturwissenschaften der Universität Regensburg

vorgelegt von

Florian Huber

aus

Ochsenhausen

im Jahr 2023

Die Arbeit entstand in gemeinsamer Betreuung durch die Universität Regensburg und die Hochschule für angewandte Wissenschaften Kempten

Gutachter und Betreuer:
Prof. Dr. Christian Wolff - Universität Regensburg
Prof. Dr. Georg Hagel - Hochschule für angewandte Wissenschaften Kempten

Florian Huber, Regensburg, 2023

Inhaltsverzeichnis

Danksagung

Einen herzlichen Dank möchte ich meinen Betreuern und Gutachtern Prof. Dr. Christian Wolff und Prof. Dr. Georg Hagel aussprechen. Sie haben mich im Rahmen der Arbeit tatkräftig unterstützt und mit wertvollem Rat begleitet. Durch die offenen Gespräch und Diskussionen durfte ich viele Dinge lernen, für die ich sehr dankbar bin.

Zudem möchte ich mich bei meinen Kolleg*innen der Hochschule Kempten sowie den Promovierenden der Universität Regensburg für all die fachlichen Gespräche, Rückfragen und Bemerkungen bedanken. Sie haben dazu beigetragen, diese Arbeit stetig zu verbessern und mir neue Perspektiven aufgezeigt. Weiterhin gilt mein Dank den Teilnehmenden der Studien respektive Evaluationen für ihr Mitwirken und ihren wertvollen Input.

Von Herzen möchte ich mich bei meiner Familie für ihre bedingungslose Unterstützung und Liebe bedanken, die ich über all die Jahre erfahren durfte. Ich bin froh, dass ihr mich auf meinem Weg begleitet habt und es auch weiterhin tun werdet.

Auch bei meinen Freunden möchte ich mich für all ihre Geduld, aufbauenden und lustigen Gespräche bedanken.

Abstract

This thesis deals with the conception and prototypical implementation of a computer-based system for teaching UML class diagrams to support the higher education of software engineering. For this, first theoretical basics are discussed, which serve as a starting point for all further developments. This is followed by an analysis of existing and possibly related work in order to distinguish it from the present work. After a comprehensive requirements analysis, the focus is on the planning and implementation of central software components. Subsequently, these components are combined to form an overall system. Finally, this is evaluated with students in a quantitative study.

Kurzzusammenfassung

Die vorliegende Arbeit befasst sich mit der Konzeption und prototypischen Implementierung eines computergestützten Systems zur Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering. Hierfür werden zuerst theoretische Grundlagen erörtert, die als Ausgangsbasis aller weiteren Entwicklungen dienen. Anschließend erfolgt eine Analyse bestehender und ggf. verwandter Arbeiten, um diese von dem vorliegenden Vorhaben abzugrenzen. Nach einer umfassenden Anforderungsanalyse steht die Planung und Umsetzung zentraler Softwarekomponenten im Fokus. Anschließend erfolgt eine Zusammenführung dieser zu einem Gesamtsystem. In einer quantitativen Untersuchung wird dieses schließlich mit Studierenden evaluiert.

Publikationen

Dieses Kapitel beinhaltet eine Auflistung der Publikationen, an denen der Autor beteiligt war. Ideen, Vorgehensweisen und weitere Vorarbeiten finden Einzug in diese Arbeit. Die entsprechenden Stellen sind durch Quellenverweise kenntlich gemacht.

- F. Huber and G. Hagel, „Work-in-Progress: Towards detection and syntactical analysis in UML class diagrams for software engineering education“, 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, 2020, pp. 3-7, https://doi.org/10.1109/EDUCON45650.2020.9125244.

- M. Müller-Amthor, G. Hagel, M. Gensheimer and F. Huber, „Scrum Higher Education – The Scrum Master Supports as Solution-focused Coach“, 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, 2020, pp. 948-952, https://doi.org/10.1109/EDUCON45650.2020.9125304.

- M. Gensheimer, F. Huber, G. Hagel, „Gamification in Software Engineering Education through Visual Novels“, Proceedings of the 4th European Conference on Software Engineering Education, Seeon/Bavaria, Germany, 2020, pp. 1-5, https://doi.org/10.1145/3396802.3396808.

- F. Huber and G. Hagel, „The application of continuous practices in higher computer science education - A systematic literature review“,

2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 2021, pp. 1618-1623, https://doi.org/10.23919/MIPRO52101.2021.9597101.

- F. Huber and G. Hagel, „Semi-automatic generation of textual exercises for software engineering education“, 2022 IEEE Global Engineering Education Conference (EDUCON), Tunis, Tunisia, 2022, pp. 51-56, https://doi.org/10.1109/EDUCON52537.2022.9766802.

- F. Huber and G. Hagel, „Tool-supported teaching of UML diagrams in software engineering education - A systematic literature review“, 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 2022, pp. 1404-1409, https://doi.org/10.23919/MIPRO55190.2022.9803560.

- F. Huber and G. Hagel, „Work-In-Progress: Converting textual software engineering class diagram exercises to UML models“, 2022 IEEE Global Engineering Education Conference (EDUCON), Tunis, Tunisia, 2022, pp. 1-3, https://doi.org/10.1109/EDUCON52537.2022.9766593.

- T. Eigler, F. Huber, and G. Hagel, „Tool-Based Software Engineering Education for Software Design Patterns and Software Architecture Patterns - a Systematic Literature Review“, Proceedings of the 5th European Conference on Software Engineering Education (ECSEE '23), Association for Computing Machinery, New York, NY, USA, 2023, pp. 153–161, https://doi.org/10.1145/3593663.3593670.

- F. Huber, T. Eigler, G. Hagel, and C. Wolff. „From Difficulties to Functional Requirements - Deriving Requirements from Literature about Tool-supported Teaching of UML Diagrams in Software Engineering Education“, Proceedings of the 5th European Conference on Software Engineering Education (ECSEE '23), Association for Computing Machinery, New York, NY, USA, 2023, pp. 184–188, https://doi.org/10.1145/3593663.3593672.

- F. Huber, T. Eigler, G. Hagel, and C. Wolff. „Qualitative Requirements Elicitation of Student Requirements for Tool-supported Tea-

ching of UML Diagrams“, Proceedings of the 5th European Conference on Software Engineering Education (ECSEE ’23), Association for Computing Machinery, New York, NY, USA, 2023, pp. 189–193, https://doi.org/10.1145/3593663.3593673.

KAPITEL 1

Einführung

> Die Hochschullehre rückt zunehmend in den Fokus bildungspolitischer Bemühungen, hochschulischer Organisationsprozesse und wissenschaftlicher Diskurse. Im Mittelpunkt steht hierbei die Verbesserung der Lehrqualität als Anpassung der Hochschulen an aktuelle politische, demografische, gesellschaftliche und wirtschaftliche Anforderungen. (Jütte u. a., 2017, S. 1)

Das vorangegangene Zitat verdeutlicht, dass eine qualitativ hochwertige Hochschullehre zunehmend in den Fokus bildungspolitischer Aktivitäten rückt. Diese stellen einen entscheidenden Faktor in der Aus- und Weiterbildung von Studierenden dar. Das gilt auch für den Fachbereich Informatik und damit korrelierende Studienfächer. Die vom Wandel geprägte, von immer schnelleren Entwicklungszyklen gezeichnete und in allen Lebensbereichen gegenwärtige Disziplin bietet große politische, gesellschaftliche aber auch wirtschaftliche Potenziale sowie Herausforderungen. Damit einher geht die Notwendigkeit einer umfassenden (hochschulischen) Ausbildung angehender Informatiker*innen, die sie auf die hohen Anforderungen ihrer beruflichen Praxis vorbereitet.

1.1 Problemkontext und der wissenschaftliche Beitrag

Die zunehmende Digitalisierung aller Lebensbereiche geht auch mit einer zunehmenden Abhängigkeit von zuverlässig arbeitenden Softwaresystemen einher (vgl. Mishra u. a., 2007). Das Hinzufügen immer umfassenderer Funktionalitäten führt zu einer merklichen Komplexitätssteigerung (vgl. Lee, 2013; vgl. Huber und Hagel, 2022b) und ist ohne strukturierte Planung und Entwicklung kaum realisierbar. Das Studienfach Software Engineering befasst sich mit diesen Inhalten und ist daher Bestandteil der meisten nationalen sowie internationalen Informatikstudiengänge (vgl. Huber und Hagel, 2022b).

In diesem Rahmen stehen Studierende häufig vor in der Literatur vielfach beschriebenen und in der Lehrpraxis wiederkehrenden Problemen. Dozierende stellen textuelle Anforderungsbeschreibungen bereit, anhand derer ein Softwaremodell oder -design erstellt werden soll. Diese Aufgabe ist für Studierende jedoch sehr herausfordernd (vgl. Stikkolorum u. a., 2018; vgl. Stuurman u. a., 2016; vgl. Huber und Hagel, 2022b). Bei der Lösungserstellung gibt es Hilfestellungen, jedoch kein Patentrezept oder standardisiertes Vorgehen. Ein studentisches Softwarediagramm kann sich visuell aber auch inhaltlich von einer Musterlösung unterscheiden und dennoch korrekt sein (vgl. Huber und Hagel, 2022b). Obgleich individuelles und zeitnahes Feedback entscheidend zum Lernerfolg der Studierenden beitragen würde, ist dies aus zeitlichen Aspekten in der Lehrpraxis nur selten realisierbar (vgl. Huber und Hagel, 2022b). Dozierende müssten Lösungsdiagramme einzeln sichten und anhand einer Begründung aufzeigen, welche Diagrammbestandteile inkorrekt modelliert wurden. Gerade bei einer größeren Anzahl an Teilnehmenden in einer Übungsgruppe ist dieses Vorgehen nicht praktikabel. Daher werden üblicherweise Musterlösungen von Dozierenden bereitgestellt. Den Studierenden ist es selbst überlassen herauszufinden, an welcher Stelle ihnen ein Fehler unterlaufen ist, warum es sich bei ihrem Diagramm nicht um eine korrekte Lösung handelt und wie sie ihre Leistung in Zukunft steigern können. Da sich eine studentische Lösung von einer bereitgestellten Musterlösung unterscheiden und dennoch korrekt sein kann, handelt es sich dabei um diffizile Aufgaben für Novizinnen und Novizen (vgl. Huber und

Hagel, 2022b). Aufgrund der Tatsache, dass sie sich in der Lernphase befinden, kann es vorkommen, dass falsche aber für richtig befundene Lösungen verinnerlicht werden.

Ein weiteres, in der Lehrpraxis bestehendes Problem stellt die Verfügbarkeit von Übungsbeispielen mit zugehörigen Musterlösungen dar (vgl. Huber und Hagel, 2022a). Die Erstellung ist für Dozierende zeitaufwendig (vgl. Huber und Hagel, 2022a; vgl. Huber und Hagel, 2022c). Daher ist das Übungsangebot für Studierende oftmals überschaubar. Die genannten Punkte sind merkliche Katalysatoren für eine Dissonanz zwischen den Anforderungen aus der Praxis (beispielsweise seitens Unternehmen) und den Fertigkeiten, die Studierende während ihres Studiums erwerben (vgl. Rodrigues u. a., 2016).

Für die Bereitstellung individueller Hilfestellungen bei der Planung und Entwicklung von Softwarediagrammen gibt es in der Literatur beschriebene Ansätze. Diese decken jedoch oftmals nicht den gesamten Übungskontext ab und unterstützen Studierende daher nur bedingt bei der Erreichung ihrer Lernziele (vgl. Huber und Hagel, 2022b). Das Ziel dieser Arbeit ist die Konzeption und prototypische Implementierung eines computergestützten Systems zur Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering. Anhand dieser Zielsetzung kann eine konkrete Forschungsfrage abgeleitet werden, die als zentrales Forschungsdesiderat der Arbeit fungiert:

> **Wie kann ein computergestütztes System zur Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering konzipiert und prototypisch implementiert werden?**

Die zentrale Forschungsfrage ist in drei Unterfragen gegliedert:

1.1 Wie kann eine Software zur Generierung textueller Aufgaben zu UML-Klassendiagrammen sowie zugehöriger Musterlösungen konzipiert und prototypisch implementiert werden?

1.2 Wie kann eine Software für den Vergleich studentischer UML-Klassendiagramme mit automatisch generierten Musterlösungen konzipiert und prototypisch implementiert werden?

1.3 In welcher Form kann Studierenden automatisiert und computergestützt individuelles und effektives Feedback zu erstellten UML-Klassendiagrammen bereitgestellt werden?

Das beschriebene Forschungsvorhaben kann als interdisziplinäres Bestreben interpretiert werden, da es Anwendung in der hochschulischen Lehre von Informatikstudiengängen und speziell in Software Engineering-Kursen findet. Daraus resultiert, dass nicht nur technische, sondern auch didaktische Aspekte bei der Konzeption berücksichtigt werden müssen. Der Fokus bleibt jedoch auf der technischen und prototypischen Umsetzung. Diese kann als Grundlage für weitere Forschungsarbeiten oder vergleichbare Zielsetzungen dienen.

Die wissenschaftliche Zielsetzung dieser Arbeit kann folgendermaßen zusammengefasst werden:

- **Tool-gestützte Lehre im Fachbereich Software Engineering:** Untersuchung zu den bereits bestehenden und in der Literatur beschriebenen Ansätzen zur Tool-gestützten Lehre von Software Engineering. Es wird herausgearbeitet, welche Funktionalitäten diese anbieten, welche Ziele damit verfolgt werden und welche Anforderungen sich daraus für ein neues System ergeben.

- **Anforderungsanalyse der Hauptakteure:** Wissenschaftlich fundierte Befragung der Zielgruppen zu ihren jeweiligen Anforderungen an ein zu entwickelndes System.

- **Konzeption und prototypische Implementierung einer Software zur Generierung textueller Aufgaben zu UML-Klassendiagrammen mit zugehörigen Musterlösungen:** Untersuchung der technischen Umsetzbarkeit einer solchen Software. Erstellung eines Prototyps anhand der Konzeption.

- **Konzeption und prototypische Implementierung einer Software für den Vergleich studentischer UML-Klassendiagramme mit einer automatisch generierten Musterlösung:** Untersuchung der technischen Umsetzbarkeit einer solchen Software. Entwicklung eines Prototyps anhand der Konzeption.

- **Konzeption eines Vorgehensmodells, das Studierenden individuelles und effektives Feedback zu erstellten UML-Klassendiagrammen bereitstellt:** Auswahl eines in der Literatur beschriebenen und anerkannten Feedback-Modells. Konzeption eines eigenen Feedback-Modells in Anlehnung an das zuvor ausgewählte.

- **Konzeption und prototypische Implementierung eines Gesamtmodells**: Zusammenführung der bisherigen Ergebnisse sowie Konzeption eines Gesamtmodells und die prototypische Implementierung dieses.

- **Evaluation:** Evaluation der einzelnen Softwarekomponenten und des Gesamtmodells mithilfe geeigneter wissenschaftlicher Methodiken und Werkzeuge.

1.2 Forschungsdesign

Das Forschungsdesign dieser Arbeit orientiert sich an dem von Hevner u. a. (2004) veröffentlichten Design Science Research (DSR) Ansatz. Laut Hevner und Chatterjee (2010) nahm dieser einen bedeutenden Einfluss auf das Information Systems Research (ISR) Forschungsgebiet (im deutschsprachigen Raum auch unter dem Begriff „Wirtschaftsinformatik“ bekannt (vgl. Wilde und Hess, 2007)).

Hevner u. a. (2004) vereinen in ihrem Ansatz zwei Paradigmen miteinander. Zum einen das konstruktionsorientierte (Design Science) und zum anderen das verhaltensorientierte (Behavioral Science) Paradigma. Sie können folgendermaßen definiert werden:

> The behavioral-science paradigm seeks to develop and verify theories that explain or predict human or organizational behavior. The design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts. (Hevner u. a., 2004, S. 75)

Demnach hat der verhaltensorientierte Ansatz das Ziel, Theorien zu entwickeln, mit denen sich das Verhalten von unternehmensspezifischen oder

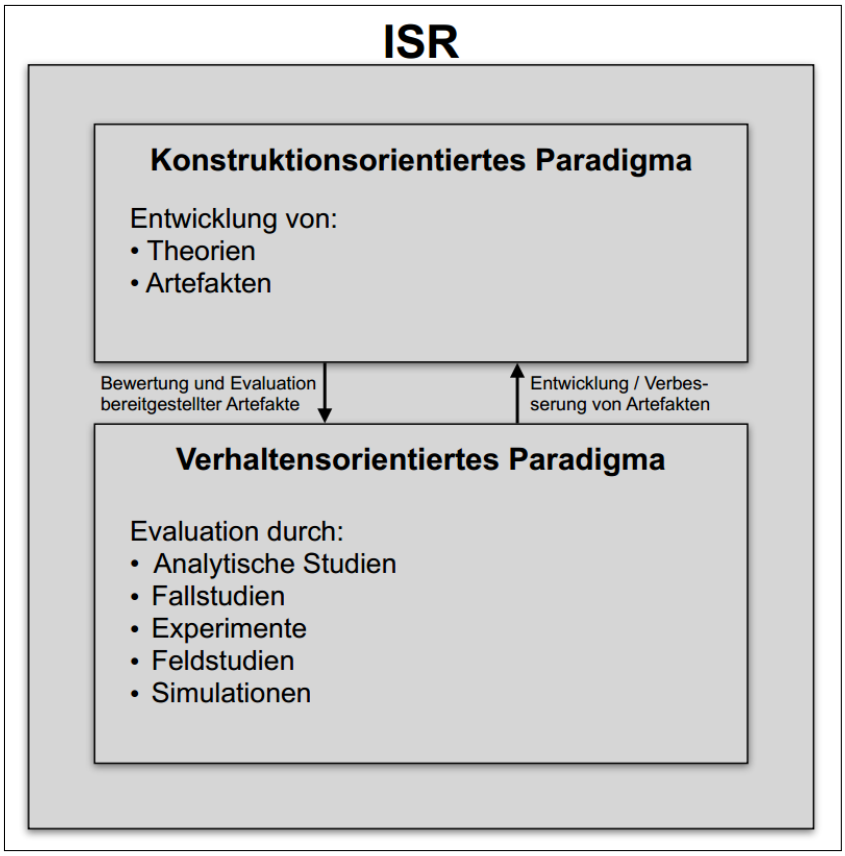


Abbildung 1.1: Zusammenhang zwischen Design- und Behavioral Science. Eigene Darstellung nach (Hevner und Chatterjee, 2010, S. 11; Hevner u. a., 2004, S. 80).

personenbezogenen Phänomenen erklären und vorhersagen lassen (vgl. Hevner u. a., 2004). Das konstruktionsorientierte Paradigma hingegen erarbeitet Theorien und Artefakte (z.B. Ideen, Praktiken, Produkte), mit deren Hilfe Probleme innerhalb von Organisationen gelöst werden sollen (vgl. Hevner u. a., 2004; vgl. Denning, 1997; vgl. Tsichritzis, 1997). Obwohl es sich um unterschiedliche Paradigmen handelt, bedingen sie einander (vgl. Hevner u. a., 2004). Dies wird in Abbildung 1.1 verdeutlicht. Das konstruktionsorientierte Paradigma stellt die entwickelten oder verbesserten Theorien und Artefakte bereit, das verhaltensorientierte bewertet und evaluiert diese unter Zuhilfenahme geeigneter Verfahren (beispielsweise analytische Studien, Experimente, Simulationen).

Des Weiteren erarbeiteten Hevner u. a. (2004) ein konzeptuelles Framework für das Verständnis, die Durchführung und die Evaluation von ISR in dem die bereits beschriebenen Paradigmen miteinander vereint werden.

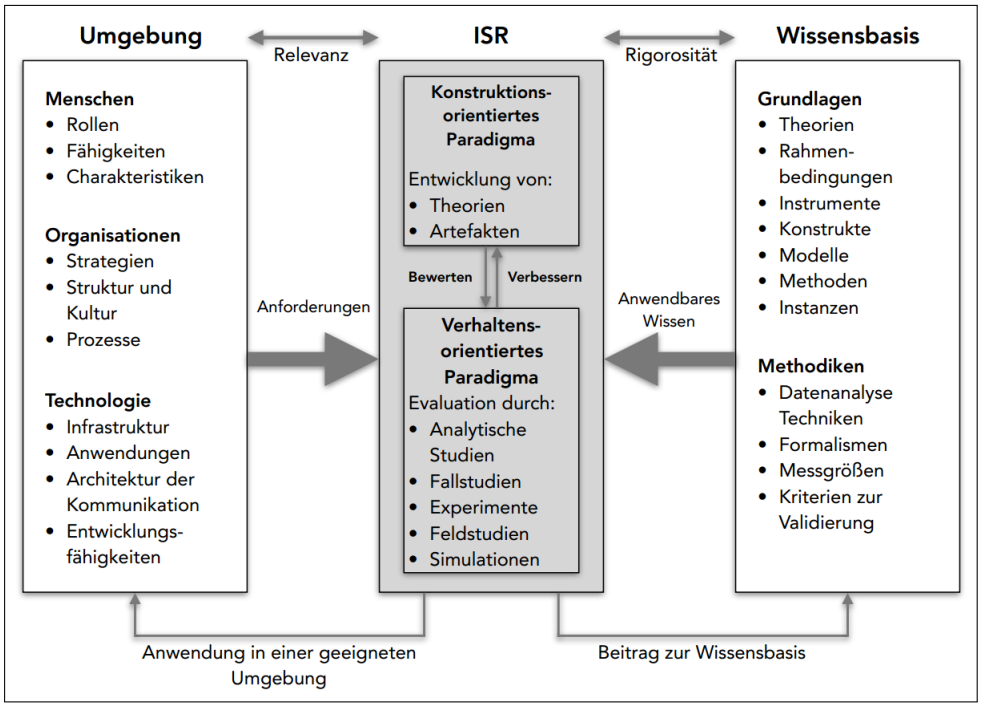


Abbildung 1.2: Das Information Systems Research Framework. Eigene Darstellung nach (Hevner u. a., 2004, S. 80).

Abbildung 1.2 stellt dieses Framework in Anlehnung an (Hevner u. a., 2004, S. 80) visuell dar. Es besteht aus drei Säulen. Die erste Säule stellt die Umgebung dar, welche sich aus Menschen, Organisationen und Technologien zusammensetzt. Außerdem beinhaltet das Framework die Säule der bestehenden Wissensbasis, die aus Grundlagenwissen und Methodiken zusammengesetzt ist. Beide Säulen wirken auf die dritte Säule, die ISR-Säule, ein. Die in Abbildung 1.1 dargestellten Paradigmen finden sich hier wieder. In dieser Säule entwickelte und evaluierte Theorien und Artefakte leisten einen Beitrag zu der bestehenden Wissensbasis und / oder finden Anwendung in einer geeigneten Umgebung. Zwischen den Säulen lässt sich ein zyklisches Verhalten identifizieren (vgl. Hevner, 2007).

Der sogenannte Relevanz-Zyklus (Übersetzung durch den Autor) verbindet die Umgebung mit ISR. Dieser Zyklus beschreibt, dass neben Anforderungen auch Akzeptanzkriterien von der Umgebung an das ISR gestellt werden. Die daraus resultierenden Theorien und Artefakte lassen sich anschließend in einer adäquaten Umgebung evaluieren. Neben der Erfüllung der Akzeptanzkriterien ist zu prüfen, inwiefern sie Verbesserungen innerhalb der Umgebung schaffen (vgl. Hevner, 2007). Ein weiterer Zyklus befin-

det sich zwischen der Säule Wissensbasis und ISR. Der Rigorositäts-Zyklus (Übersetzung durch den Autor) stellt Grundlagen und Methodiken bereit, die von Forschenden durch eine geschickte Auswahl und Anwendung zur Konstruktion und Bewertung von Theorien und Artefakten beitragen. Deren Entwicklung dient wiederum als Erweiterung der Wissensbasis und kann als Grundlage für weitere und neue Forschungsansätze dienen (vgl. Hevner, 2007).

Neben dem soeben beschriebenen Framework stellen Hevner u. a. (2004) sieben Richtlinien bereit, die sich bei der Entwicklung von Theorien und Artefakten als Grundlage heranziehen lassen. Tabelle 1.1 fasst diese Richtlinien zusammen. Das Forschungsdesign der Arbeit orientiert sich an dem beschriebenen Framework und den vorgestellten Richtlinien. Demnach erfolgt die Konzeption und prototypische Implementierung eines Artefakts respektive einer Softwareanwendung für die Unterstützung der hochschulischen Lehre von UML-Klassendiagrammen (siehe Richtlinie R1), das eine technologiebasierte Lösung für die in Abschnitt 1.1 relevante Problemstellung darstellt (siehe Richtlinie R2). Untersuchungen zur Güte, Wirksamkeit und dem Nutzen des Artefakts sind durch geeignete Verfahrensweisen respektive Evaluationsmethoden erfolgt (siehe Richtlinie R3). Eine Beschreibung dieser findet sich in den folgenden Kapiteln. Das Vorgehen stützt sich demnach sowohl auf die Entwicklung, als auch die Bewertung des Artefakts (siehe Richtlinie R4). Dafür kommen unter Berücksichtigung der Umweltbedingungen geeignete Technologien und Methodiken zum Einsatz (siehe Richtlinie R5). Sowohl die Funktionsweise des Artefakts als auch die Resultate der Evaluationen dienen als Grundlage für Forschungsbeiträge, die wiederum eine Erweiterung der Wissensbasis darstellen (siehe Richtlinie R6 und R7).

1.3 Struktur der Arbeit

Die Struktur der Arbeit ergibt sich anhand des zuvor beschriebenen Forschungsdesigns. In der Umgebung auftretende Herausforderungen, Problemstellungen sowie Anforderungen werden beobachtet und dienen als Grundlage für die Entwicklung eines Artefakts. In dieser Arbeit handelt es sich dabei um ein konzipiertes und prototypisch implementiertes Softwaresystem

Richtlinie		Beschreibung
ID	**Name**	
R1	Design als Artefakt	DSR muss eine brauchbare Theorie oder ein brauchbares Artefakt in Form eines Konstrukts, Modells etc. hervorbringen.
R2	Problemrelevanz	Ziel der DSR ist die Entwicklung technologiebasierter Lösungen für relevante Probleme.
R3	Design-Auswertung	Nutzen, Qualität und Wirksamkeit der Theorie / des Artefakts müssen durch geeignete Evaluationsmethoden nachgewiesen werden.
R4	Rigorosität der Forschung	DSR stützt sich sowohl bei der Entwicklung als auch bei der Bewertung von Theorien / Artefakten auf strenge, definierte Methoden.
R5	Design als Forschungsprozess	Die Suche nach einer wirksamen Theorie / eines wirksamen Artefakts erfordert den Einsatz verfügbarer Mittel um das gewünschte Ziel unter Berücksichtigung der Umweltbedingungen zu erreichen.
R6	Forschungsbeiträge	DSR muss nachweisbare und nachvollziehbare Beiträge im Forschungsbereich der Theorie / des Artefakts liefern.
R7	Präsentation der Forschungsergebnisse	DSR muss effektiv präsentiert werden. Dies gilt sowohl für technologisch orientierte, als auch für weitere Stakeholder.

Tabelle 1.1: Richtlinien für die Erstellung von Theorien und Artefakten. Eigene Darstellung nach (Hevner u. a., 2004, S. 83).

zur Unterstützung der Lehre von UML-Klassendiagrammen in der hochschulischen Lehre von Software Engineering. Das Artefakt stützt sich dabei auf unterschiedliche Deep-Learning-Technologien (besteht jedoch nicht ausschließlich daraus). Die theoretische Basis dieser Entwicklung stellt die bestehende Wissensbasis dar. In den in Abbildung 1.2 dargestellten Zyklen wird das Artefakt mithilfe dieses Wissens entwickelt und anschließend in der Umgebung evaluiert. Die daraus resultierenden Ergebnisse tragen zur Weiterentwicklung der Wissensbasis bei.

Die Kapitel der Arbeit sind wie folgt strukturiert:

Kapitel 1: Einführung
Das erste Kapitel beschreibt den vorliegenden Problemkontext und den wissenschaftlichen Beitrag der Arbeit. Weiterhin werden Forschungsdesiderate und -fragen sowie das Forschungsdesign erörtert.

Kapitel 2: Theoretische Grundlagen
Das zweite Kapitel bietet einen Überblick über relevantes Hintergrundwissen. Die darin beschriebenen Terminologien, Theorien und Methodiken finden im Laufe der Arbeit Anwendung.

Kapitel 3: Anforderungserhebung
Das dritte Kapitel extrahiert Anforderungen, die als Entwicklungsgrundlage für das Artefakt dienen. Dabei wird zum einen die bestehende Wissensbasis untersucht. An Stellen, an der diese keine hinreichenden Antworten liefern kann, werden zum anderen Anforderungen innerhalb der Umgebung erhoben. Es resultiert eine umfassende Liste gewünschter Funktionalitäten, die das Artefakt bereitstellen sollte.

Kapitel 4: Generierung textueller Übungsaufgaben und zugehöriger Musterlösungen
Das vierte Kapitel befasst sich mit der automatisierten Erstellung textueller Übungsaufgaben und zugehöriger Musterlösungen. Ziel ist somit die Beantwortung von Forschungsfrage 1.1. Es beschreibt das Vorgehen und ordnet dieses in das gewählte Forschungsdesign ein. Nach einer Zusammen-

fassung verwandter Arbeiten erfolgt eine Beschreibung der durch den Autor durchgeführten Vorarbeit. Anschließend werden jeweils Softwarelösungen für die automatisierte Erstellung textueller Übungsaufgaben sowie zugehöriger Musterlösungen konzipiert, prototypisch implementiert und schließlich evaluiert.

Kapitel 5: Vergleich studentischer Lösungen mit automatisch generierten Musterlösungen
Das fünfte Kapitel beschreibt, wie studentische UML-Klassendiagramme (studentische Lösungen zu vorliegenden Übungsaufgaben) mit einer automatisiert erzeugten Musterlösung verglichen werden können. Auch hier findet sich eine Beschreibung des Vorgehens und eine Einordnung in das Forschungsdesign, gefolgt von einer Zusammenfassung verwandter Arbeiten und der durch den Autor durchgeführten Vorarbeit. Weiterhin wird eine Softwarelösung konzeptionell geplant und prototypisch implementiert. Nach einer Evaluation lässt sich Forschungsfrage 1.2 beantworten.

Kapitel 6: Feedback
Das sechste Kapitel ist vergleichbar strukturiert, wie die vorherigen beiden Kapitel. Darin wird ein anhand der Wissensbasis abgeleitetes Feedback-Modell konzipiert und prototypisch implementiert. Die Resultate ermöglichen die Beantwortung der Forschungsfrage 1.3.

Kapitel 7: Konzeption und prototypische Implementierung
Das siebte Kapitel befasst sich mit der Beantwortung der zentralen Forschungsfrage. Dafür wird das in diesem Kapitel beschriebene Vorgehen in das Forschungsdesign eingeordnet, bevor eine konzeptionelle Planung eines umfänglichen und computergestützten Systems für die Lehre von UML-Klassendiagrammen erfolgt. Neben einer Erläuterung der prototypischen Implementierung ist weiterhin eine Darstellung der Benutzerschnittstelle in Form einer interaktiven Webseite enthalten.

Kapitel 8: Evaluation
Das achte Kapitel evaluiert das Gesamtsystem mithilfe einer quantitativen Untersuchung in Form eines Fragebogens. Damit werden Studierende zu ih-

rer Einschätzung und Bewertung des Prototyps befragt.

Kapitel 9: Zusammenfassung und Ausblick
Das neunte und letzte Kapitel fasst die Arbeit in ihren Schwerpunkten zusammen und erläutert die wissenschaftlichen Beiträge. Darüber hinaus erfolgt ein Ausblick über mögliche zukünftige Forschungsarbeiten und Entwicklungen in diesem Kontext.

KAPITEL 2

Theoretische Grundlagen

Dem DSR-Forschungsdesign nach Hevner u. a. (2004) zufolge, muss in einem ersten Schritt die bestehende Wissensbasis analysiert werden. Daher erfolgt in diesem Kapitel eine Erläuterung der grundlegenden Konzepte, die für das Verständnis dieser Arbeit vonnöten sind. Diese Wissensbasis dient, ganz im Sinne des DSR-Ansatzes, als Grundlage und Ausgangspunkt für weitere Entwicklungen.

Zunächst werden die Fachdisziplin Software Engineering und die in diesem Rahmen häufig verwendete Unified Modeling Language (UML) vorgestellt. Anschließend erfolgt eine Vorstellung von Herausforderungen in diesen Fachdisziplinen. Daraufhin wird der Begriff Deep Learning mitsamt prominenter Modellarchitekturen erläutert und drei für diese Arbeit relevante Modelle vorgestellt. Letztlich wird das Thema Feedback näher beleuchtet.

2.1 Software Engineering Education

Bei einer NATO-Konferenz im Jahr 1968 wurde der Begriff Software Engineering erstmals geprägt (vgl. Lee, 2013; vgl. Krusche u. a., 2018). Dieser befasst sich mit dem Phänomen, dass viele Softwareprojekte festgelegte Budget- und Zeitvorgaben nicht einhalten konnten, mit einer schlechten Qualität bereitgestellt oder gar nicht erst beendet wurden (vgl. Lee, 2013).

Das Phänomen wird in der Literatur oftmals mit dem Begriff „software crisis“ (zu deutsch „Software-Krise“) referenziert. Um diesen Problemen entgegenzuwirken, empfahlen die Konferenzteilnehmenden den Fokus nicht rein auf die Programmierung, sondern auf den gesamten Planungs- und Herstellungsprozess von Software zu legen (vgl. Lee, 2013).

Das IEEE Standard Glossary of Software Engineering Terminology definiert den Begriff Software Engineering folgendermaßen:

> [...] application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; [...]. („IEEE Standard Glossary of Software Engineering Terminology“ 1990, S. 67)

Es handelt sich demnach um einen systematisierten Ansatz zur Planung, Entwicklung, dem Betrieb sowie der Wartung von Software, der den von der NATO-Konferenz vorgeschlagenen Lösungsansatz für die Software-Krise in sich trägt. Die Aktivitäten und Tätigkeitsfelder dieser Disziplin sind beispielsweise in den ISO-Normen 15288-2015 (vgl. „ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes“ 2015) oder 24748-2-2018 (vgl. „ISO/IEC/IEEE International Standard - Systems and Software Engineering– Life Cycle Management– Part 2: Guidelines for the Application of ISO/IEC/IEEE 15288 (System Life Cycle Processes)“ 2018) sowie im Software Engineering Body of Knowledge (SWEBOK) (vgl. Bourque und Fairley, 2014) nachzulesen. Abbildung 2.1 visualisiert die in den ISO-Normen beschriebenen technischen Schritte zur Analyse, Planung, Erstellung und Bereitstellung von Software. Alle Inhalte wurden durch den Autor in die deutsche Sprache übersetzt. Da eine detaillierte Beschreibung aller einzusehenden Prozesse sehr umfangreich ausfallen würde, sei an dieser Stelle auf die Originalquellen verwiesen.

Heute, über 50 Jahre nach der NATO-Konferenz, ist eine umfassende, strukturierte und vorausschauende Planung sowie Entwicklung von Software wichtiger denn je. Durch die steigende Digitalisierung sind immer mehr Unternehmen und staatliche Einrichtungen von Softwaresystemen abhängig (vgl. Mishra u. a., 2007; vgl. Vyshnevskyi, 2020). Diese werden in ihrem Funktionsumfang zunehmend komplexer (vgl. Lee, 2013; vgl. Huber und Hagel, 2022b). Auch die oftmals notwendige Kommunikation unterschiedlicher

heterogener Softwaresysteme untereinander erschwert die Softwareentwicklung zusätzlich. Die Aufgabe eines Software Engineers ist es, die Planung und Entwicklung dieser komplexen Umgebungen effektiv zu verwalten (vgl. Mishra u. a., 2007).

Daher ist es wenig verwunderlich, dass die meisten Universitäten und Hochschulen für angewandte Wissenschaften in ihren Informatikstudiengängen (oder damit korrelierenden Studiengängen) Software Engineering-Kurse anbieten, um die Studierenden auf ihre beruflichen Herausforderungen vorzubereiten (vgl. Huber und Hagel, 2022b). Der Begriff Software Engineering Education beschreibt in diesem Kontext die Lehre von Software Engineering-Inhalten, die ein Verständnis über die Konzepte sowie Prinzipien dieser Fachdisziplin schaffen und Kompetenzen zur Lösung realer Probleme vermitteln soll (vgl. Ouhbi und Pombo, 2020).

Gerade das Design von Softwarekomponenten stellt Studierende in Software Engineering-Kursen regelmäßig vor Herausforderungen (vgl. Stikkolorum u. a., 2018; Stuurman u. a., 2016; Huber und Hagel, 2022b). Oftmals erhalten Studierende eine unstrukturierte textuelle Anforderungsbeschreibung, die sie in ein Diagramm überführen sollen (vgl. Huber und Hagel, 2022a; Huber und Hagel, 2022b). Die Texte sollen Kunden-, Projekt- oder Stakeholderanforderungen darstellen, mit denen sich im Bereich Informatik arbeitende Personen in ihrer beruflichen Praxis konfrontiert sehen. Dieser Lehrkontext ist in Abbildung 2.1 durch die Schritte „Erfassung und Definition von (Stakeholder-)Anforderungen“ und „Planung und Erstellung des Softwaredesigns“ sowie einer Verbindung dieser Prozesse visualisiert. Dazwischenliegende Schritte werden in einem hochschulischen Kontext oftmals übersprungen, um den Umfang der Aufgabenstellung gering zu halten.

Die Beschreibung und Dokumentation geplanter, bestehender und noch zu entwickelnder Softwaresysteme und -komponenten ist eine der zentralen Aufgaben von Software Engineering. In akademischen aber auch unternehmensspezifischen Kontexten findet dafür häufig die UML Anwendung.

Im Rahmen dieser Arbeit erfolgt eine Abgrenzung der Begrifflichkeiten ‚textuelle Anforderungsbeschreibung‘, ‚Aufgabenstellung‘ und ‚Übungsaufgabe‘ zueinander. Erstere verweist auf eine textbasierte Beschreibung von Diagrammkomponenten, deren Inhalte (in Form von Attributen und Metho-

den) sowie deren Beziehungen untereinander. Sie sind natürlichsprachlich verfasst und folgen keiner stringenten Struktur. Die Aufgabenstellung hingegen gibt vor, welche Schritte für die Bearbeitung einer textuellen Anforderungsbeschreibung durchzuführen sind. In einem hochschulischen Umfeld ist die Vorlage beider Komponenten in Kombination sinnvoll und hier durch den Begriff der Übungsaufgabe definiert. Die Lehre von Software Engineering kann eine Vielzahl weiterer Aufgabentypen beinhalten, die hier jedoch keine Berücksichtigung finden.

2.1.1 Die Unified Modeling Language

Um die bereits im vorherigen Kapitel erwähnte steigende Komplexität von Software beherrschbarer zu machen, entwickelten sich seit den 1960er Jahren objektorientierte Konzepte und Programmiersprachen, die teils auch heute noch Anwendung finden (vgl. Seidl u. a., 2015). Diese gliedern Software in abstrakte Artefakte, auch Objekte genannt, die Informationen verwalten sowie Operationen ausführen können (vgl. Lee, 2013). Aus menschlicher Sicht ist das ein sehr intuitiver Ansatz, da er gut mit unserem Weltbild vereinbar ist (vgl. Seidl u. a., 2015). Mithilfe von Modellierungssprachen (in der Literatur auch mit dem englischen Begriff „Modeling Language“ referenziert) lassen sich die unterschiedlichen Objekte einer Software und deren Beziehung zueinander visualisieren. Ihre Entwicklung begann Mitte der 1970er Jahre aufgrund der steigenden Komplexität von Softwaresystemen, die mit objektorientierten Programmiersprachen entwickelt wurden (vgl. Booch u. a., 1998). Zwischen den Jahren 1989 und 1994 vervielfachte sich die Anzahl beschriebener Modellierungssprachen von zehn auf über 50 (vgl. Booch u. a., 1998; vgl. Bézivin und Muller, 1999). Software Ingenieur*innen fiel es zunehmend schwer, eine einheitlich Kommunikationsbasis zu entdecken, die zusätzlich auch noch all ihren Anforderungen an eine Modellierungssprache gerecht wird.

Zu den beliebtesten Vertretern gehörten damals Object-Oriented Software Engineering (OOSE) von Ivar Jacobson, Booch von Grady Booch und Object Modeling Technique (OMT) von James Rumbaugh, die alle ihre individuellen Stärken und Schwächen aufwiesen (vgl. Booch u. a., 1998). Bei der Weiterentwicklung ihrer Ansätze adaptierten sie Ideen voneinander

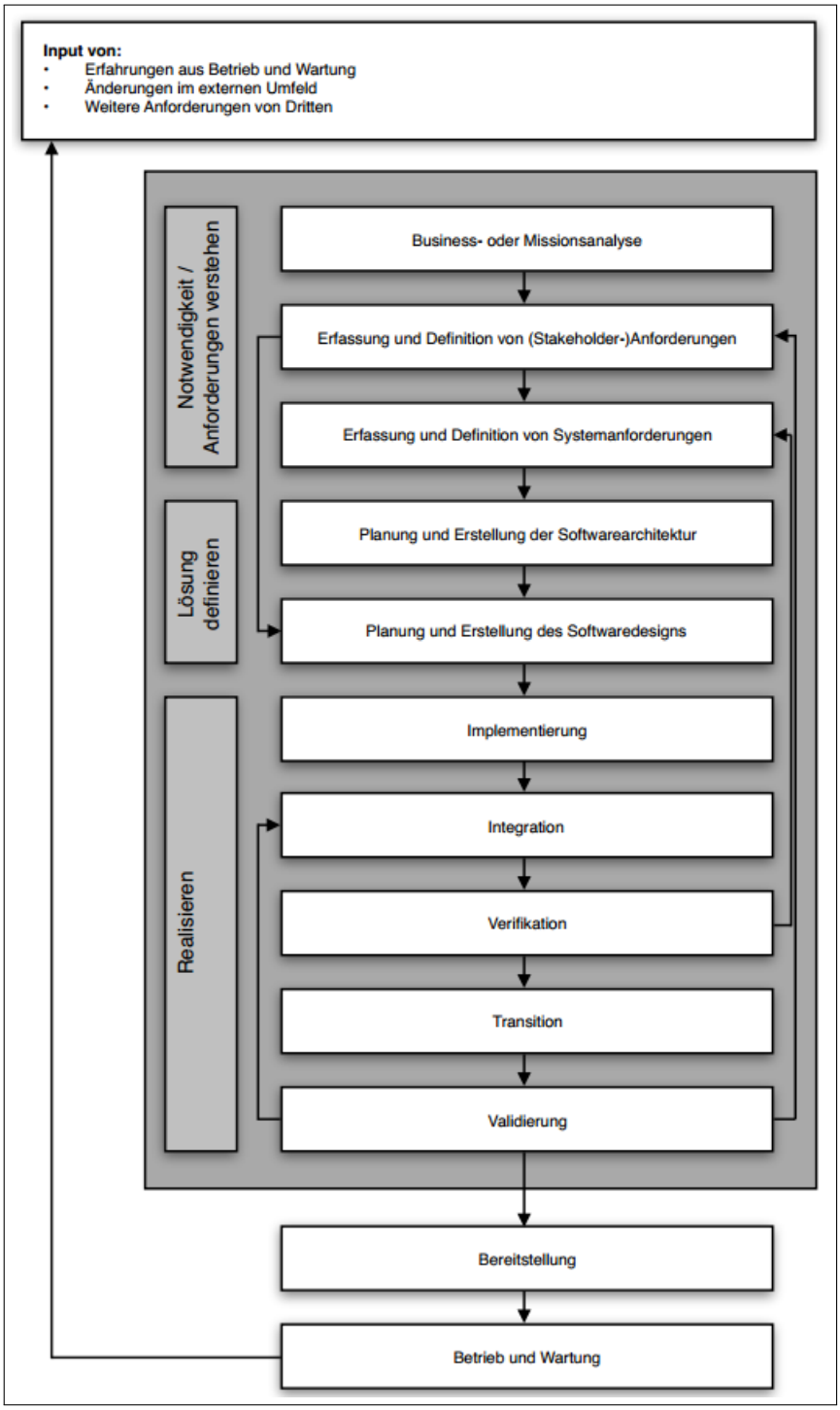


Abbildung 2.1: Technische Software Engineering-Prozesse in der Analyse, Planung, Erstellung und Bereitstellung von Software. Eigene Darstellung nach („ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes“ 2015, S. 11-104; „ISO/IEC/IEEE International Standard - Systems and Software Engineering– Life Cycle Management– Part 2: Guidelines for the Application of ISO/IEC/IEEE 15288 (System Life Cycle Processes)“ 2018, S. 39f.).

und vereinten im Jahr 1994 schließlich ihre Bemühungen um eine standardisierte Modellierungssprache zu schaffen, die heute als UML bekannt ist (vgl. Booch u. a., 1998; vgl. Bézivin und Muller, 1999; vgl. Seidl u. a., 2015) und als industrieller sowie akademischer Standard (siehe ISO/IEC 19505-1:2012[1]) gilt, wenn es um die Planung und das Design von Software geht. Die Inhalte dieses Kapitels lassen sich mit den Worten von Seidl (2015) wie folgt zusammenfassen:

> The Unified Modeling Language (UML) is a consolidation of the best practices that have been established over the years in the use of modeling languages. UML enables us to present the widely varying aspects of a software system (e.g., requirements, data structures, data flows, and information flows) within a single framework using object-oriented concepts. (Seidl u. a., 2015, S. 1)

Die UML stellt für unterschiedliche Anwendungsfälle und Szenarien verschiedene Modelltypen bereit (vgl. Oestereich und Bremer, 2009). Das wohl am häufigsten verwendete ist das UML-Klassendiagramm (vgl. Seidl u. a., 2015). Daher liegt der Fokus dieser Arbeit auf diesem Modelltyp. Alle Konzepte lassen sich jedoch auch auf andere UML-Diagramme übertragen. Für eine genaue Beschreibung aller Elemente, die in einem UML-Klassendiagramm enthalten sein können, sei auf Oestereich und Bremer (2009) sowie Seidl u. a. (2015) verwiesen.

Mithilfe der UML ist ein breites Spektrum von Lösungsmöglichkeiten für gegebene, sich teils wiederholende Problemstellungen modellierbar. Die Effektivität der entwickelten Resultate kann dabei variieren. Für Software Ingenieur*innen kann die Neuentwicklung von Lösungen zu bereits bekannten Herausforderungen viel Zeit in Anspruch nehmen. Deutlich effizienterer hingegen ist es, eine erprobte Vorgehensweise respektive Schablone heranzuziehen und diese auf die vorliegenden Sachverhalte anzuwenden (vgl. Gamma u. a., 1994). Besagte Schablonen sind auch unter dem Begriff ‚Entwurfsmuster' bekannt, mit denen sich der folgende Unterabschnitt befasst.

[1] https://www.iso.org/standard/32624.html

2.1.2 Entwurfsmuster

Ein erstelltes Softwaredesign muss verschiedenen Ansprüchen gerecht werden. So sollte es spezifisch genug sein, um gegebene Problemstellungen zu lösen, aber gleichzeitig generalisierbar und für neue Anforderungen erweiterbar bleiben (vgl. Gamma u. a., 1994). Auch mit Erfahrung ist das eine herausfordernde Aufgabe. Bei der Modellierung von Softwaresystemen und -komponenten treten wiederkehrende Herausforderungen auf. Sogenannte Entwurfsmuster (engl. „Design Patterns“) helfen dabei, diese mithilfe eines definierten, abstrakten und verifizierten Vorgehens zu lösen (vgl. Gamma u. a., 1994; vgl. Heer und Agrawala, 2006). Die Prägung des Begriffs führen Gamma u. a. (1994) auf Alexander u. a. (1977) zurück. Dessen Fokus lag ursprünglich auf Mustern bei der Planung und dem Bau von Gebäuden und Städten, ist jedoch auch auf die Softwareentwicklung anwendbar (vgl. Alexander u. a., 1977; vgl. Gamma u. a., 1994).

Entwurfsmuster sind unabhängig von Programmiersprachen und während der Modellierung auf das individuelle Problem anzupassen. Daraus resultierende Lösungen sind in der Regel nicht identisch, folgen jedoch demselben Grundprinzip. Nach Gamma u. a. (1994) gibt es vier zentrale Elemente, die ein Entwurfsmuster beinhalten muss (eigene Übersetzung nach (vgl. Gamma u. a., 1994, S. 3)):

1. Name: Aus Gründen der Identifizierbarkeit und Differenzierbarkeit erhält jedes Entwurfsmuster eine individuelle Benennung.

2. Problembeschreibung: Ist eine Kurzzusammenfassung des wiederkehrenden Problems und beschreibt, unter welchen Bedingungen das Entwurfsmuster einsetzbar ist.

3. Lösung: Ist eine Beschreibung von Komponenten, deren Beziehungen und Verantwortlichkeiten, mit denen die Problemstellung lösbar ist. Sie sind nicht spezifisch sondern bewusst abstrakt gehalten, da das Entwurfsmuster als anpassbare Schablone fungiert.

4. Konsequenzen: Sind eine Zusammenfassung der aus der Anwendung des Entwurfsmusters resultierenden Effekte und Abwägungen.

Entwurfsmuster stellen eine große Erleichterung bei der Modellierung dar, da durch ihre Anwendung das sprichwörtliche Rad nicht immer wieder neu

erfunden werden muss. Aufgrund ihrer Relevanz in der Praxis sind sie häufig Bestandteil der hochschulischen Lehre von Software Engineering.

2.1.3 Herausforderungen für Studierende bei der Modellierung mithilfe der UML im hochschulischen Kontext

In der hochschulischen Lehre gibt es eine Vielzahl an Problematiken, mit denen sich Studierende konfrontiert sehen. Diese müssen nicht zwangsläufig Bezug zu bestimmten Lehrinhalten aufweisen. Unabhängige Gründe für das unvollständige oder fehlerhafte Bearbeiten einer Aufgabe können beispielsweise mangelnde Vorkenntnisse oder geringe intrinsische Motivation sein (vgl. Jonassen, 1999). Diese stellen keinen Bestandteil der Untersuchungen sowie Fragestellungen dieser Arbeit dar.

Vielmehr sollen in diesem Abschnitt Herausforderungen für Studierende angeführt werden, die jeweils im Kontext der hochschulischen Lehre von Software Engineering sowie UML-Klassendiagrammen auftreten. Diese sind in der folgenden Auflistung zusammengefasst, wurden aus der bestehenden Fachliteratur entnommen und sind durch das frei gewählte Kürzel „HLA“ sowie eine eindeutige Nummer identifizierbar:

HLA1 Ingenieursdisziplinen (zu denen auch Software Engineering zählt) können üblicherweise nicht in Isolation voneinander gelernt und betrachtet werden. Dies hat zur Folge, dass Studierende deren Grundlagen unabhängig des jeweiligen Spezialgebiets meistern müssen (vgl. Ghezzi und Mandrioli, 2006).

HLA2 Im Software Engineering sind viele, aber nicht alle, Prinzipien anderer Ingenieursdisziplinen transferierbar, da Software ein immaterielles Produkt und üblicherweise nicht frei von Fehlern ist (vgl. Bracken, 2003).

HLA3 Software Engineering ist eine vergleichsweise junge Ingenieursdisziplin, in der theoretische Grundlagen und etablierte Modelle noch nicht vollständig ausgereift sein können (vgl. Bracken, 2003; vgl. Ghezzi und Mandrioli, 2006; vgl. Oguz und Oguz, 2019).

HLA4 Unterschiedliche Softwareprojekte und deren Anforderungen sind üblicherweise nicht identisch (vgl. Bracken, 2003). Studierende müssen lernen, wiederkehrende Probleme zu identifizieren und bekanntes Wissen individuell zu adaptieren (vgl. Ghezzi und Mandrioli, 2006).

HLA5 Im Übungsbetrieb mangelt es oftmals an realistischen Rahmenbedingungen. Studierende können nach Beendigung einer Aufgabe damit abschließen und erkennen selten die Konsequenzen ihrer Designentscheidungen (vgl. Bracken, 2003).

HLA6 Die Aufgabenstellungen in den Software Engineering-Übungen sind in hochschulischen Kontexten meist wohl definiert und klar abgegrenzt (vgl. Bracken, 2003). Dies ist in der Realität selten der Fall.

HLA7 Studierende müssen erlernen, mit der Mehrdeutigkeit und teils auch Widersprüchlichkeit von Anforderungen umzugehen (vgl. Bracken, 2003).

HLA8 Wie bei realen Softwareprojekten müssen Studierende manchmal in Gruppen zusammenarbeiten (vgl. Bracken, 2003). Dies kann sich aufgrund unterschiedlicher Wissensstände, Motivationen und kultureller Hintergründe als herausfordernd gestalten (vgl. Bracken, 2003; Ghezzi und Mandrioli, 2006).

HLA9 Neben dem Erwerb theoretischen Wissens ist es für Studierende wichtig, auch ein großes Spektrum praktischer Erfahrungen zu sammeln (vgl. Ghezzi und Mandrioli, 2006). Aus zeitlichen Gründen kann dies jedoch häufig nur eingeschränkt stattfinden.

HLA10 Studierende müssen lernen, Software so komplex wie nötig und so simpel wie möglich zu gestalten (vgl. Ghezzi und Mandrioli, 2006).

Neben den bereits angeführten Herausforderungen stoßen Studierende auch auf spezifische Schwierigkeiten, die in der hochschulischen Lehre von UML-Klassendiagrammen auftreten können (identifizierbar durch das frei gewählte Kürzel „HLK“ sowie eine eindeutige Nummer):

HLK1 Das Erlernen und praktische Umsetzen objektorientierter Konzepte stellt eine große Herausforderung für Studierende dar (vgl. Thomasson

u. a., 2006; vgl. Or-Bach und Lavy, 2004; vgl. Reuter u. a., 2020; vgl. Stikkolorum u. a., 2018).

HLK2 Bei der Unterscheidung sowie Anwendung von Assoziationen treten häufig Verwechslungen auf (vgl. Reuter u. a., 2020; vgl. Kruus u. a., 2014). So fällt es Studierenden beispielsweise schwer, Aggregationen von Kompositionen zu differenzieren.

HLK3 Die Modellierung aller in einer Anforderungsbeschreibung genannten Elemente zur Herstellung korrekter Zusammenhänge ist für Studierende eine Herausforderung (vgl. Reuter u. a., 2020).

HLK4 Die Auswahl der korrekten UML-Komponente zur Darstellung eines beschriebenen Sachverhalts gelingt Studierenden nicht immer (vgl. Huber und Hagel, 2022a).

HLK5 Das Modellieren nicht funktionaler Anforderungen ist in UML-Klassendiagrammen nicht üblich. Es kommt jedoch vor, dass Studierende versuchen, diese dem Diagramm hinzuzufügen (vgl. Reuter u. a., 2020).

HLK6 Der Verwendungszweck und die richtige Anwendung von Multiplizitäten sowie Kardinalitäten ist Studierenden nicht immer bewusst (vgl. Reuter u. a., 2020; vgl. Bolloju und Leung, 2006).

HLK7 Es kommt vor, dass Studierende fälschlicherweise Implementierungsdetails in ihr UML-Klassendiagramm einfügen (vgl. Chren u. a., 2019).

HLK8 Da Studierende unterschiedliche Wissensstände aufweisen, ist die Bearbeitung einer Übungsaufgabe für einige Studierende eine größere und zeitlich intensivere Herausforderung (vgl. Duschl u. a., 2014).

HLK9 Die Übertragung des in den Köpfen der Studierenden bestehenden mentalen Modells in ein UML-Klassendiagramm gelingt in vielen Fällen nicht fehlerfrei (vgl. Duschl u. a., 2014).

HLK10 Die korrekte Interpretation und das Nachvollziehen einer Aufgabenstellung ist für Studierende nicht immer möglich (vgl. Duschl u. a., 2014; vgl. Stikkolorum u. a., 2018).

HLK11 Das nicht vollständige Lesen der Aufgabenstellung kann zu Verwirrungen und Fehlern in einem UML-Klassendiagramm führen (vgl. Duschl u. a., 2014). Auch eine Fülle unterschiedlich beschriebener Komponenten kann diesen Effekt erzielen (vgl. Duschl u. a., 2014).

HLK12 Diverse Lernstrategien und Arbeitsweisen von Studierenden führen zu qualitativ unterschiedlichen Ergebnissen (vgl. Duschl u. a., 2014; vgl. Huber und Hagel, 2020).

HLK13 Die praktische Anwendung gelernter, theoretischer Inhalte stellt eine Hürde für Studierende dar (vgl. Huber und Hagel, 2020).

HLK14 Aufgrund eines meist begrenzten Pools an Übungsaufgaben, fehlt vielen Studierenden die praktische Erfahrung, um neue und komplexe Aufgabenstellungen lösen zu können (vgl. Huber und Hagel, 2022c; vgl. Huber und Hagel, 2022a).

HLK15 Die Verwendung von Modellierungstools kann Studierenden hinsichtlich der Komplexität dieser Softwareanwendung bei der Diagrammerstellung hindern (vgl. Stikkolorum u. a., 2018).

HLK16 Die gewünschte Granularität eines UML-Klassendiagramms zu erkennen und zu realisieren, ist für Studierende oftmals nicht möglich (vgl. Stikkolorum u. a., 2018).

HLK17 Assoziationen zwischen Klassen werden von Studierenden häufig nicht korrekt oder überhaupt nicht erkannt (vgl. Chren u. a., 2019; vgl. Stikkolorum u. a., 2018). Es kann vorkommen, dass diese zwar identifiziert, aber schlichtweg bei der Modellierung vergessen werden (vgl. Chren u. a., 2019).

HLK18 Studierenden ist in manchen Fällen unklar, wie die Richtung einer gerichteten Assoziation zu bestimmen ist (vgl. Reuter u. a., 2020).

HLK19 Die Entscheidung Attribute oder Assoziationen einzusetzen, stellt Studierende bei Modellierungen vor Herausforderungen (vgl. Reuter u. a., 2020).

HLK20 Es können Unklarheiten darüber auftreten, ob eine Information als Klasse, Attribut oder Methode zu modellieren ist (vgl. Reuter u. a., 2020; vgl. Stikkolorum u. a., 2018).

HLK21 Studierende können sich in den Anforderungsbeschreibungen wiederholende Wörter fälschlicherweise als Klasse identifizieren (vgl. Reuter u. a., 2020).

HLK22 Mit einer Vielzahl an in den Anforderungsbeschreibungen enthaltenen Komponenten fällt es Studierenden häufig schwer, diese gänzlich zu detektieren und zu modellieren (vgl. Thomasson u. a., 2006; vgl. Huber und Hagel, 2020; vgl. Stikkolorum u. a., 2018).

HLK23 Es können von Studierenden keinerlei Relationen zu weiteren Diagrammbestandteilen hergeleitet werden, auch wenn Klassen identifiziert wurden (vgl. Thomasson u. a., 2006).

HLK24 In einigen Fällen finden Studierende keine sinnvollen Bezeichnungen für modellierte Attribute (vgl. Reuter u. a., 2020).

HLK25 Die korrekte Zuordnung bestimmter Attribute zu ihren jeweiligen Klassen ist für Studierende nicht immer ersichtlich (vgl. Bolloju und Leung, 2006).

HLK26 Studierenden fällt das Modellieren von Methoden in Klassen schwer (vgl. Reuter u. a., 2020).

HLK27 Gelegentlich modellieren Studierende Methoden in Klassen, denen sie nicht angehören (vgl. Chren u. a., 2019; vgl. Bolloju und Leung, 2006).

HLK28 Studierende erkennen Methoden nicht als solche oder vergessen sie in ihren Modellierungen (vgl. Chren u. a., 2019; vgl. Stikkolorum u. a., 2018).

HLK29 Das korrekte Hinzufügen von Übergabeparametern in Methoden kann eine Schwierigkeit darstellen (vgl. Chren u. a., 2019).

HLK30 Es kann vorkommen, dass Studierende den Rückgabetyp von Methoden vergessen oder diesen nicht korrekt klassifizieren (vgl. Chren u. a., 2019).

Beide Aufzählungen fassen eine Fülle auftretender Herausforderungen für Studierende in den genannten Themengebieten zusammen. Die große Anzahl nicht gelöster Problem- bzw. Fragestellungen verdeutlicht das Potenzial und die akademische sowie praktische Relevanz dieses Forschungsvorhabens.

2.2 Künstliche Neuronale Netze

Ein Künstliches Neuronales Netz (KNN) versucht, biologische Strukturen nachzubilden und damit komplexe Problemstellungen zu bewältigen. Dieser Abschnitt beleuchtet prägnant einige theoretische Grundlagen und definiert relevante Begriffe, bevor prominente Netzarchitekturen vorgestellt werden. Weiterhin sind konkrete Modelle beschrieben. Die Inhalte des Abschnitts dienen als Ausgangspunkt für die weiteren Entwicklungen der Arbeit und folglich der Beantwortung der Forschungsfragen. So lassen sich besagte Modelle beispielsweise heranziehen, um bei der Generierung von Übungsaufgaben sowie zugehöriger Musterlösungen oder bei einem visuellen Import von UML-Klassendiagrammen zu unterstützen.

2.2.1 Hinführung

Unter den Säugetieren zählt das menschliche Gehirn als kognitiv leistungsfähigstes (vgl. Azevedo u. a., 2009) und ist bis dato auch den Computern voraus, wenn es um das Erlernen neuer Sachverhalte und das Zurechtfinden in unbekannten Situationen sowie Umgebungen geht. Es enthält durchschnittlich 86,1 ($\pm$ 8,1) Milliarden Neuronen (vgl. Azevedo u. a., 2009). Dabei handelt es sich um vernetzte Nervenzellen, die elektrische aber auch chemische Signale verarbeiten und ggf. weiterleiten können (vgl. Dörn, 2018). Erreichen die empfangenen, elektrischen Spannungsimpulse einen Schwellwert θ im Zellkörper eines Neurons, wird ein Spannungsimpuls weitergeleitet (vgl. Dörn, 2018). Dieser kann durch chemische Vorgänge gehemmt oder verstärkt werden (vgl. Du und Swamy, 2014). Die Vernetzung mehrerer Neurone miteinander wird als neuronales Netz bezeichnet (vgl. Dörn, 2018).

Der Wunsch, diese extrem leistungsfähige, biologische Struktur technisch nachzubilden und für unterschiedliche Anwendungsgebiete nutzbar zu ma-

chen, besteht in der Informatik schon lange. Bereits im Jahr 1943 entwickelten der Neuropsychologe Warren McCulloch und der Mathematiker Walter Pitts ein erstes künstliches Neuron, das auf elektrischen Schaltkreisen basierte (vgl. McCulloch und Pitts, 1943; vgl. Wang u. a., 2017). Für eine einfache Berechnungseinheit stellten sie mehrere Eingangs- und einen einzelnen Ausgangskanal bereit (vgl. McCulloch und Pitts, 1943). Die zugrundeliegende mathematische Funktion summiert die gewichteten Eingangssignale und gibt den Wert 1 zurück, sofern ein Schwellwert θ überschritten wurde (vgl. McCulloch und Pitts, 1943; vgl. Wang u. a., 2017). Andernfalls gab sie den Wert 0 zurück (vgl. McCulloch und Pitts, 1943; vgl. Wang u. a., 2017). Der Ansatz von McCulloch und Pitts (1943) erlaubte folglich eine binäre Klassifikation von Eingangsparametern. Die Gewichtungen wurden vorab durch einen Menschen mathematisch bestimmt und waren durch das Modell nicht selbst erlernbar (vgl. Goodfellow u. a., 2016).

Im Jahr 1949 beschrieb Donald Hebb, der heute als Vater der Neuronalen Netze gilt (vgl. Didier und Bigand, 2011), wie sich biologische, neuronale Strukturen durch Lernprozesse adaptieren können. Das folgende Zitat fasst diese Annahme prägnant zusammen:

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. (Hebb, 1949, S. 62)

Obwohl Hebb (1949) seine Arbeit im Hinblick auf biologische und nicht auf technischen Strukturen verfasste, ist der Grundgedanke lernender Berechnungseinheiten übertragbar.

Inspiriert durch die Arbeit von McCulloch und Pitts (1943) sowie Hebb (1949) veröffentlichten Frank Rosenblatt, Alexander Stieber und Robert Schatz bereits 1957 einen Ansatz für ein mathematisches, technisches Neuron, das sogenannte Perzeptron (vgl. Rosenblatt u. a., 1957). Es stellt bis heute die Grundlage für technische Neuronenmodelle dar. Abbildung 2.2 zeigt ein exemplarisches Perzeptron. Das Modell besteht aus Eingabeneuronen, die mit einem Ausgabeneuron verbunden sind.

Der Eingabevektor x setzt sich aus den Eingabewerten x_1 bis x_n zusam-

men (vgl. Dörn, 2018). Der Gewichtsvektor aus den zu den Eingabewerten gehörenden Gewichten w_1 bis w_n (vgl. Dörn, 2018). Die Eingabe E des Ausgabeneurons besteht aus dem Skalarprodukt des Eingabevektors x und dem Gewichtsvektor w (vgl. Dörn, 2018):

$$E = \sum_{i=1}^{n} x_i w_i$$

In Abbildung 2.2 ist dies auf der linken Seite des Ausgabeneurons mit einem Summensymbol kenntlich gemacht. Ähnlich wie bei seinem biologischen Vorbild leitet das Perzeptron Signale nur dann weiter, wenn ein Schwellwert θ überschritten wird. Dieser ist üblicherweise durch eine Eingabe von $x_0 = 1$ und ein Gewicht $w_0 = -\theta$ abgebildet (vgl. Dörn, 2018).

Das Ergebnis des Skalarprodukts der Vektoren x und w wird in eine Aktivierungsfunktion $f_\theta(E)$ gegeben (in Abbildung 2.2 auf der rechten Seite des Ausgabeneurons durch ein f gekennzeichnet). Für die Ausgabe y des Ausgabeneurons gilt (vgl. Dörn, 2018; vgl. Walde, 2005):

$$y = f_\theta(E) = \begin{cases} 1, & E = \sum_{i=1}^{n} x_i w_i - \theta = \sum_{i=0}^{n} x_i w_i > 0 \\ 0, & \text{sonst} \end{cases}$$

Anhand von Trainingsdaten können die Gewichte w_0 bis w_n eingestellt und somit bestimmte Resultate für bestimmte Eingabeparameter gelernt werden (vgl. Dörn, 2018). Das in Abbildung 2.2 gezeigte herkömmliche Perzeptron, diente der linearen Separierung zweier Mengen und konnte dadurch boolsche mathematische Funktionen wie beispielsweise die UND-Funktion $f_{UND} = x_1 \wedge x_2$ sowie die ODER-Funktion $f_{ODER} = x_1 \vee x_2$ abbilden (vgl. Dörn, 2018).

Die angeführten Entwicklungen zwischen den 1940er und 1960er Jahren sind heute nicht als Deep Learning sondern unter dem Begriff Cybernetics (im deutschsprachigen Raum „Kybernetik“ genannt) oder auch als “erste Welle“ in der Entwicklung künstlicher neuronaler Strukturen bekannt. Bis dato konnten diese Ansätze nur linear separierbare Datensätze klassifizieren und stießen dadurch häufig an Grenzen.

Das menschliche Gehirn legt Daten nicht wie ein Computer an bestimmten Speicherstellen ab, um sie bei Bedarf abzurufen (vgl. Beck u. a., 2018). Vielmehr besitzt jede Information ein spezifisches Aktivierungsmuster im menschlichen neuronalen Netz (vgl. Beck u. a., 2018; vgl. Dörn,

2018). Nimmt das menschliche Gehirn neue Sinneseindrücke wahr, passt es die Verbindungen zwischen Neuronen an, womit das Aktivierungsmuster in Zukunft leichter aktivierbar ist - das Gehirn lernt (vgl. Beck u. a., 2018). Auch der von Hebb (1949) beschriebene Grundgedanke findet sich in diesen Erkenntnissen wieder und führte zwischen 1980 und 1995 zu einer „zweiten Welle“ in der Entwicklung künstlicher Neuronenstrukturen, die heute auch unter dem Begriff Connectionsism (im deutschsprachigen Raum „Konnektionsimus“ genannt) bekannt ist (vgl. Rumelhart u. a., 1986b; vgl. McClelland u. a., 1986; vgl. Goodfellow u. a., 2016). Mit der Idee der Schaffung *intelligenten Verhaltens* (Goodfellow u. a., 2016, S. 16) wurden einzelne technische, neuronale Berechnungseinheiten miteinander verknüpft und mit einer nicht-linearen Sigmoid-Funktion als Aktivierungsfunktion versehen (vgl. Dörn, 2018; vgl. Walde, 2005). Ein sogenanntes KNN ist in eine Eingabeschicht (Input-Layer), eine variable Anzahl von Zwischenschichten (Hidden-Layer) und eine Ausgabeschicht (Output-Layer) unterteilt (vgl. Dörn, 2018; vgl. Pattanayak, 2017). Der Zusammenschluss mehrerer Perzeptrone innerhalb einer solchen Netzwerkarchitektur wird als Multi Layer Perzeptron (MLP) bezeichnet. Sie sind in der Lage, auch nicht-linear separierbare Datensätze zu klassifizieren (sofern nicht-lineare Aktivierungsfunktionen zum Einsatz kommen).

Als weitere große Errungenschaft der zweiten Welle galt die Entwicklung eines Algorithmus, mit dessen Hilfe die Gewichte einzelner Neurone in einem KNN erfolgreich und vergleichbar effizient einstellbar waren (vgl. Rumelhart u. a., 1986a; vgl. LeCun und Fogelman-Soulié, 1987; vgl. Goodfellow u. a., 2016).

Bis in die Mitte der 1990er Jahre gab es ein großes Aufsehen und viele unrealistische Erwartungen an die KNN (vgl. Goodfellow u. a., 2016). Als diese nicht erfüllt und zunehmend bessere Machine Learning Modelle entwickelt wurden, ebbte die Begeisterung für Neuronale Netze vorerst ab (vgl. Goodfellow u. a., 2016).

Die „dritte Welle“ in der Entwicklung dieser Netzstrukturen wird seit dem Jahr 2006 unter dem Begriff Deep Learning geführt. Die Verfügbarkeit zunehmend leistungsstarker Hardware, eine große Anzahl vorhandener Trainingsdaten sowie neue Algorithmen respektive Netzstrukturen führten

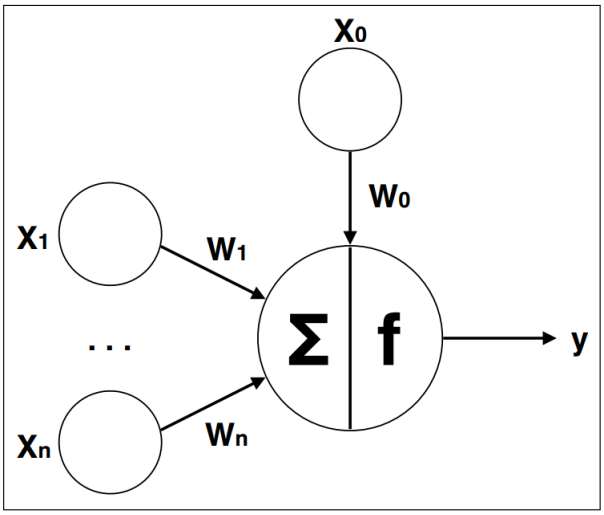


Abbildung 2.2: Das Perzeptron. Eigene Darstellung in Anlehnung an (Pattanayak, 2017, S. 90).

in den vergangenen Jahren zu großen Erfolgen und wachsender Begeisterung für diese Modelle (vgl. Plebe und Grasso, 2019). Eine genaue Definition dieses Begriffs findet sich im folgenden Unterabschnitt.

2.2.2 Der Begriff Deep Learning

Deep Learning wird im Kontext dieser Arbeit wie folgt definiert:

> This solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all the knowledge that the computer needs. The hierarchy of concepts enables the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning. (Goodfellow u. a., 2016, S. 1f.)

Durch die Verwendung tiefer Architekturen mit vielen Neuronen in den einzelnen Schichten können diese Modelle sehr abstrakte Informationen lernen und sind somit in vielen Anwendungsbereichen einsetzbar (vgl. Plebe und

Grasso, 2019), wie beispielsweise in der Analyse und Erzeugung von natürlicher Sprache. Die unter dem Begriff Natural Language Processing (NLP) bekannte Disziplin kann, wie folgt, definiert werden:

> Natural language processing (NLP) is a theory-motivated range of computational techniques for the automatic analysis and representation of human language. NLP enables computers to perform a wide range of natural language related tasks at all levels, ranging from parsing and part-of-speech (POS) tagging, to machine translation and dialogue systems. (Young u. a., 2018, S. 1)

Beispielhafte auf Deep Learning basierende NLP-Modelle für die Analyse von Textelementen sowie für die Generierung neuer Texte finden sich jeweils in Abschnitt 2.2.4.1 und 2.2.4.2. Zunächst erfolgt jedoch eine prägnante Erläuterung verbreiteter Deep-Learning-Architekturen.

2.2.3 Prominente Netzarchitekturen

Aufgrund ihrer mannigfaltigen Einsatzmöglichkeiten und beeindruckender Erfolge befanden sich Deep-Learning-Ansätze in den vergangenen Jahren im Zentrum vieler Forschungs- und Entwicklungsarbeiten (vgl. Plebe und Grasso, 2019). Bei den in diesem Unterabschnitt angeführten Architekturen handelt es sich um prominente Vertreter ihrer Gattung. Diese werden prägnant vorgestellt. Für eine ausführlichere Beschreibung der Funktionsweisen und technischen Details sei an dieser Stelle auf die jeweils angeführte Fachliteratur verwiesen.

2.2.3.1 Convolutional Neural Networks

Das Convolutional Neural Network (CNN) ist eine von der Biologie inspirierte Netzwerkarchitektur, die häufig für die Detektion, Lokalisation und Klassifikation von Objekten in Bildern Einsatz findet (vgl. Aggarwal, 2018). Bei diesen Aufgaben können sie die menschliche Leistungsfähigkeit übertreffen (vgl. He u. a., 2016; vgl. Montesinos López u. a., 2022).

Die Pixel eines Bildes setzen sich üblicherweise aus drei numerischen Werten zusammen. Jede dieser Zahlen repräsentiert dabei einen Farbkanal

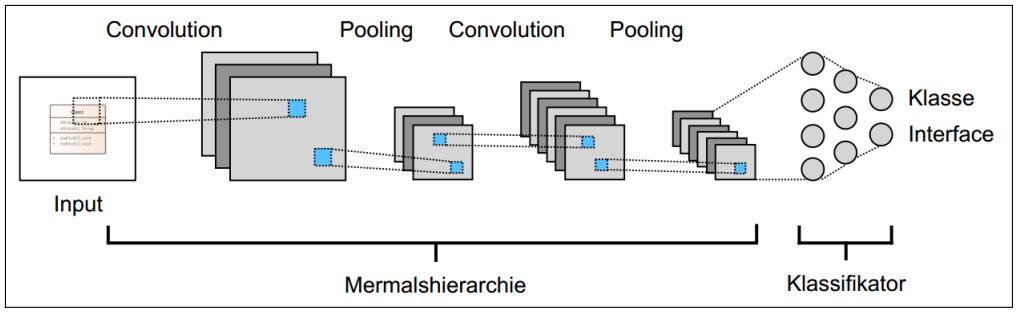


Abbildung 2.3: Beispielhafter Aufbau eines CNN. Eigene Darstellung in Anlehnung an (Aggarwal, 2018, S.41; Montesinos López u. a., 2022, S. 547).

(rot, grün oder blau). Folglich kann ein solches Bild aus technischer Perspektive als dreidimensionale Matrix interpretiert werden:

$$\textit{Bildmatrix} = \textit{Anzahl Pixel in der Breite} * \textit{Anzahl Pixel in der Höhe} * \textit{Anzahl Farbkanäle}$$

Matrizen in dieser Form dienen als Eingabeparameter für CNNs (vgl. Manaswi, 2018). Ihren Namen erhielten diese Modelle aufgrund sogenannter Faltungen, die im englischen als „Convolution“ bezeichnet werden (vgl. Montesinos López u. a., 2022). Dabei handelt es sich um eine mathematische Operation, mit deren Hilfe die Merkmale eines Bildes (beispielsweise horizontale und vertikale Kanten oder Farbvariationen) extrahierbar sind (vgl. Manaswi, 2018; vgl. Montesinos López u. a., 2022). Abbildung 2.4 visualisiert die Faltungsoperation exemplarisch. Ausschnitte der Eingabematrix werden dabei nacheinander elementweise mit einer sogenannten Filtermatrix (auch häufig als „Kernel“ bezeichnet) multipliziert und deren jeweiligen Produkte aufsummiert.

Nach jeder Rechenoperation verändert sich der betrachtete Ausschnitt der Eingabematrix. In Abbildung 2.4 würde bildlich gesprochen der in markierte Ausschnitt um eine (oder mehrere) Spalte(n) nach rechts verschoben werden. Die resultierenden Ergebnisse bilden eine neue Matrix, die in der Fachsprache auch als Feature Map referenziert ist (vgl. Manaswi, 2018). Für die Extraktion unterschiedlicher Bildmerkmale finden häufig mehrere Filter Anwendung. Somit entstehen multiple Feature Maps.

Neben der Convolution finden in CNNs auch sogenannte Pooling-Operationen Anwendung (vgl. Montesinos López u. a., 2022). Sie dienen der Dimensionsreduktion und teils auch der Rauschreduzierung innerhalb der Fea-

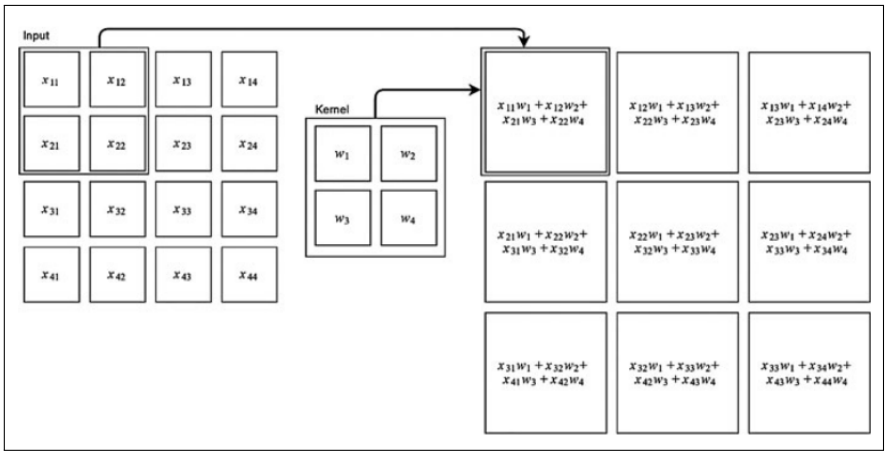


Abbildung 2.4: Darstellung einer Convolution-Operation. Die Abbildung wurde aus (Montesinos López u. a., 2022, S. 541) entnommen.

ture Maps. Das Pooling ist anhand unterschiedlicher mathematischer Funktionen durchführbar. Populär ist die Verwendung des sogenannten Max-Poolings (vgl. Montesinos López u. a., 2022). Hierbei werden Ausschnitte der Feature Map betrachtet und der maximale numerische Wert in einer neuen Feature Map gespeichert. Abbildung 2.5 visualisiert dies exemplarisch. Nach jeder Operation lässt sich der blau markierte Betrachtungsbereich um einen oder mehrere Schritte verschieben. Dieser sogenannte „stride“ ist einstellbar und hier beispielhaft auf $s = 2$ gesetzt (vgl. Montesinos López u. a., 2022).

Beide Operationen werden dabei in Form von Hidden-Layern aneinandergereiht und bilden eine Merkmalshierarchie. Die Anzahl der Schichten ist dabei variabel. Die resultierenden Feature Maps dienen als Eingabeparameter für einen Klassifikator. Bei diesem handelt es sich in der Regel um ein MLP. Abbildung 2.3 veranschaulicht die beschriebene Architektur eines CNN beispielhaft.

2.2.3.2 Rekurrente Neuronale Netze

Gegenüber dem herkömmlichen MLP ist ein Rekurrentes Neuronales Netz (RNN) besonders gut für die Klassifikation, Vorhersage und Generierung sequenzieller Daten geeignet (vgl. Manaswi, 2018). Hierzu gehören beispielsweise Texte, gesprochene Sprache und numerische Zeitreihen wie Aktienkurse. Die zugrundeliegende Netzwerkarchitektur gleicht der des MLPs. Bei Letzterem sind die Neurone eines Layers jedoch nur mit den Neuronen des jeweils nächsten Layers verknüpft. Das RNN hingegen weist zusätzliche Ver-

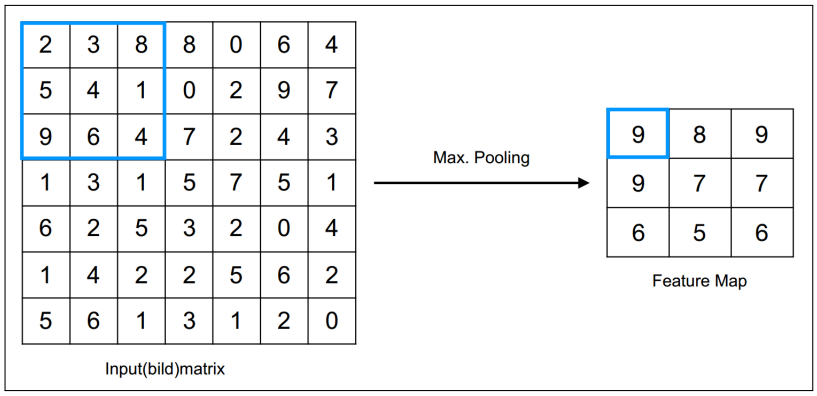


Abbildung 2.5: Exemplarische Darstellung einer Pooling-Operation. Eigene Darstellung in Anlehnung an (Montesinos López u. a., 2022, S. 543).

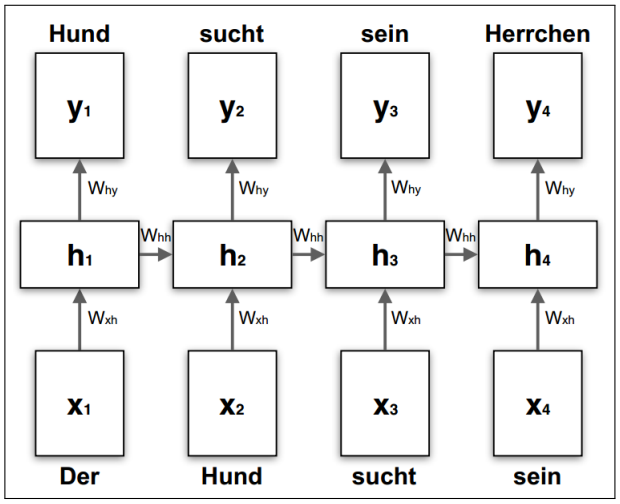


Abbildung 2.6: Exemplarische Darstellung eines RNN. Eigene Darstellung in Anlehnung an (Aggarwal, 2018, S. 39).

bindungen der Neurone innerhalb eines Layers zueinander oder zu den Neuronen vorheriger Schichten auf (vgl. Dörn, 2018). Dies ermöglicht die sequenzielle Betrachtung von Eingabewerten in Abhängigkeit eines Zeitpunkts t. Abbildung 2.6 visualisiert diesen Sachverhalt. In dem Beispiel dienen aneinandergereihte Worte als Eingabeparameter, die mit x_t beschriftet sind. Jeder Zeitschritt hat weiterhin einen Ausgabeparameter y_t. Der Zustand h_t ergibt sich aus einer Funktion f, welche den vorherigen Zustand h_{t-1} und den Eingabeparameter x_t berücksichtigt (in Anlehnung an (vgl. Aggarwal, 2018, S. 39)):

$$h_t = f(h_{t-1}, x_t)$$

Die angeführte Funktion verdeutlicht, dass Informationen aus vorherigen

Zuständen in Kombination mit einem Eingabewert zum Zeitpunkt t als Basis für die Errechnung des kommenden Ausgabeparameters dienen (vgl. Manaswi, 2018). In dem angeführten Beispiel werden dem RNN schrittweise die Worte „Der“, „Hund“, „sucht“, „sein“ vorgelegt. Durch Training ist das Netzwerk in der Lage, das Wort „Herrchen“ als nächsten Satzbaustein zu prädizieren.

Einen der bekanntesten Vertreter dieser Netzwerkarchitekturen stellt das Long Short-Term Memory (LSTM) dar (vgl. Hochreiter und Schmidhuber, 1997; vgl. Manaswi, 2018). Dabei handelt es sich um eine modifizierte Variation herkömmlicher RNNs. Eine technische Beschreibung unterschiedlicher Modellvarianten findet sich in der Fachliteratur (vgl. Greff u. a., 2017).

2.2.3.3 Autoencoder-Modelle

Autoencoder sind spezielle neuronale Netze, die gegebene Eingabeparameter in eine bedeutsame sowie dimensionsreduzierte Repräsentation komprimieren und diese anschließend mit vermindertem Informationsgehalt wieder korrekt nachbilden können (vgl. Hinton und Salakhutdinov, 2006; vgl. Bank u. a., 2020). Abbildung 2.7 visualisiert exemplarisch die Netzwerkarchitektur dieser Modelle und verdeutlicht den zuvor beschriebenen Sachverhalt. Die Komprimierung der Eingabeparameter wird mithilfe eines sogenannten Encoders durchgeführt (vgl. Hinton und Salakhutdinov, 2006). Dieser besteht aus unterschiedlichen Layern, die eine Dimensionsreduktion herbeiführen. Wie bei CNNs können hier beispielsweise Convolutional- und Pooling-Layer Anwendung finden. Der Encoder stellt die ursprünglichen Eingabeparameter in einer komprimierten Repräsentation wie einem n-dimensionalen Vektor oder einer n-dimensionalen Matrix bereit. Über einen Decoder, der ebenfalls aus unterschiedlichen Layern besteht, sind diese Informationen in ihren Ursprungszustand rückführbar. Der Informationsgehalt wird dabei jedoch reduziert, was in Abbildung 2.7 zu einer Reduktion des Rauschens in einem Bild führt. Weiterhin sind Autoencoder-Modelle auch für Klassifikationsaufgaben (die komprimierte Repräsentation wird dabei nicht dem Decoder, sondern einem Klassifikator vorgelegt (vgl. Bank u. a., 2020)) oder für die Generierung von Bildern sowie Texten geeignet.

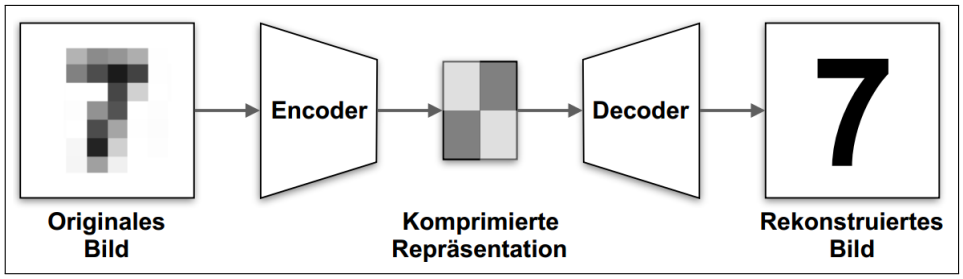


Abbildung 2.7: Exemplarische Darstellung eines Autoencoders. Eigene Darstellung in Anlehnung an (Bank u. a., 2020, S. 2).

2.2.3.4 Transformer-Modelle

Gerade bei der Verarbeitung natürlicher Sprache oder im Bereich der Computer Vision konnten in den vergangenen Jahren mithilfe sogenannter Transformer-Modelle bemerkenswerte Erfolge erzielt und bisherige Deep-Learning-Modelle in den Schatten gestellt werden (vgl. Lin u. a., 2021). So kamen bisher beispielsweise für die Generierung von Texten die in Abschnitt 2.2.3.2 beschriebenen LSTMs zum Einsatz. Aufgrund der sequenziellen Verarbeitung der Textbausteine sind diese Netzwerkarchitekturen gerade bei längeren Texten fehleranfällig (vgl. Gillioz u. a., 2020). Transformer-Modelle hingegen sind in der Lage, alle Textbausteine parallel und in Relation zueinander zu betrachten (vgl. Vaswani u. a., 2017). Dies ist mithilfe sogenannter Attention-Layer möglich (vgl. Krüger, 2021). Die Architektur besteht ebenfalls aus Encodern und Decodern, welche unter anderem die Attention-Layer beinhalten. Die Netzwerkarchitektur des ersten vorgestellten Transformer-Modells findet sich in Abbildung 2.8. Da die Komplexität der jeweiligen Komponenten einer ausführlichen Beschreibung bedarf, sei hierfür auf die bestehende Fachliteratur verwiesen (vgl. bspw. Vaswani u. a., 2017; vgl. bspw. Krüger, 2021).

2.2.4 Konkrete Beispiele für Deep-Learning-Modelle

In diesem Unterabschnitt finden sich konkrete Modelle, die auf den zuvor beschriebenen Netzarchitekturen aufbauen. Auch hier sei für technische Einzelheiten auf die Originalquellen verwiesen.

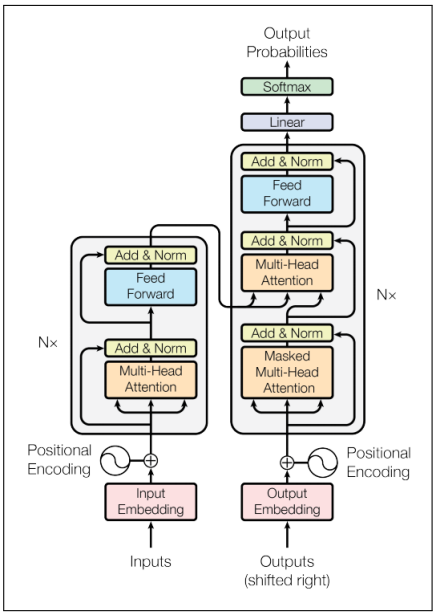


Abbildung 2.8: Die Netzwerkarchitektur eines Transformer-Modells. Die Abbildung wurde aus (Vaswani u. a., 2017, S. 3) entnommen.

2.2.4.1 Analyse von Textbausteinen

Die Möglichkeit, sehr komplexe Informationen und Zusammenhänge erlernen zu können, ermöglicht es Deep-Learning-Modellen natürliche Sprache zu verarbeiten, zu übersetzen oder zu erzeugen (vgl. Plebe und Grasso, 2019; vgl. Young u. a., 2018). Aufgrund dieser weitreichenden praktischen Anwendungsmöglichkeiten handelt es sich bei NLP um ein sehr aktives Forschungsgebiet mit einer Vielzahl von Modellen und Forschungsansätzen.

spaCy ist eine open-source, state-of-the-art Python-Bibliothek für komplexe Auswertungen von Sätzen und Textbausteinen (vgl. „*spaCy 101: Everything you need to know* 2022“; vgl. Partalidou u. a., 2019), die unter anderem auf LSTMs setzt. Mithilfe der Bibliothek kann beispielsweise die Wortart (Substantiv, Verb etc.) einzelner Satzbestandteile und die Wortabhängigkeit dieser identifiziert werden. Abbildung 2.9 veranschaulicht dies anhand eines erstellten Beispiels. Ein beliebiger Satz wurde mithilfe der Bibliothek analysiert. Die unterschiedlichen Wortarten sind unter dem zugehörigen Wort sichtbar. Die Art der Wortabhängigkeiten ist mithilfe von Pfeilen und Abkürzungen kenntlich gemacht (so hat das Nomen „Tag“ beispielsweise eine Verbindung zu dem Adjektiv „schöner“, da letzteres ein beschreibendes Wort für ersteres darstellt). Es ist anzumerken, dass in diesem

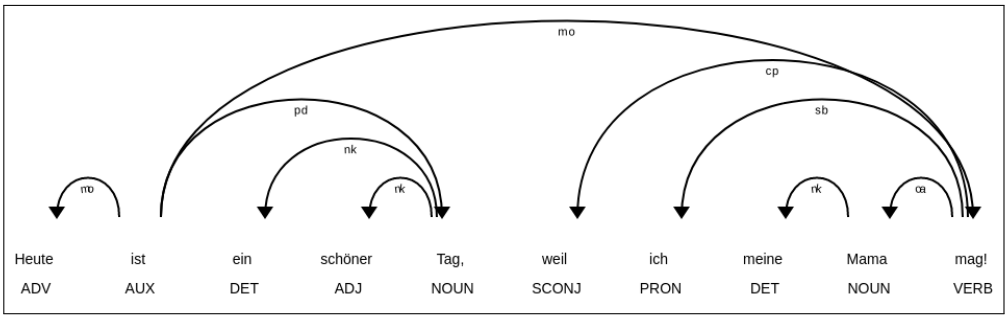


Abbildung 2.9: Analyse von Satzbestandteilen unter Verwendung der spaCy-Bibliothek.

Beispiel keine Detektion von Satzzeichen stattgefunden hat.

Eine Auflistung des umfassenden Funktionsumfangs[2] sowie eine Übersicht zugehöriger Leistungsdaten[3] finden sich auf der Webseite der Python-Bibliothek. Da diese recht umfangreich sind, wurde von einer eigenen Zusammenfassung abgesehen.

Im Folgenden soll spaCy herangezogen werden, um zu untersuchen, wie sich vorliegende textuelle Anforderungsbeschreibungen zu UML-Klassendiagrammen automatisiert in Musterlösungen überführt lassen. Die Python-Bibliothek soll dabei die Extraktion der beschriebenen Diagrammbestandteile ermöglichen. Somit dient sie als wichtiger Baustein, der zur Beantwortung von Forschungsfrage 1.1 beitragen wird.

2.2.4.2 Generierung von Text

Das von OpenAI entwickelte GPT-2 Modell ist ein sehr prominenter Vertreter der Transformer-Modelle (vgl. Radford u. a., 2019). Es wurde für unterschiedliche NLP-Aufgaben, wie beispielsweise das Zusammenfassen, Übersetzen oder Vervollständigen von Texten, entwickelt (vgl. Radford u. a., 2019). Es wurde mit über acht Millionen Dokumenten (ca. 40GB) aus dem Internet trainiert (vgl. Radford u. a., 2019). Das Modell konnte sehr gute Ergebnisse erzielen und wurde von OpenAI weiterentwickelt. Das vorgestellte GPT-3 Modell (vgl. Brown u. a., 2020) ist jedoch nicht kostenfrei verfügbar und findet daher in dieser Arbeit keine Anwendung.

Abbildung 2.10 zeigt beispielhaft, wie mithilfe eines auf die englische Sprache trainierten GPT-2 Modells Texte generiert werden können. Ein

[2]Auflistung des Funktionsumfangs: https://spacy.io/usage/spacy-101
[3]Übersicht zugehöriger Leistungsdaten: https://spacy.io/models/de

```
[INPUT]
For students in software engineering education it seems challenging to design software components.

[OUTPUT]
For students in software engineering education it seems challenging to design software components. The current trend
is to use standard libraries that can be used to solve problems, but this can create a big barrier to participation
for new students in computer science. Another problem is the requirement to install programs that are not designed a
nd written to use such software. In any case we recommend that we not rely solely on the ability of the software to
be used in computer science. Instead we should use hardware components such as CPUs or GPUs,
```

Abbildung 2.10: Generierung von Text in englischer Sprache mithilfe des GPT-2 Modells.

Satz im oberen Bereich der Abbildung dient dem Modell als Anhaltspunkt zur Generierung weiterer Sätze. Das Ergebnis (im unteren Bereich der Abbildung sichtbar) ist bis dato nicht optimal, kann aber durch eigenes Training optimiert werden. Dies ist im Kontext der Arbeit gerade im Hinblick auf die automatisierte Erstellung von Übungsaufgaben interessant. Im Folgenden soll das Modell mithilfe von erstelltem Trainingsmaterial auf diese Aufgabe spezifiziert werden. Somit stellt es einen relevanten Bestandteil bei der Beantwortung von Forschungsfrage 1.1 dar.

2.2.4.3 Detektion und Lokalisation von Objekten in Bildern

Das ursprünglich von Redmon u. a. (2016) entwickelte You Only Look Once (YOLO) Modell zählt zu den bekanntesten Echtzeit-Objektdetektoren. Das Deep-Learning-Modell verarbeitet eingehende Bilder in einer einzelnen Prozedur und ist in der Lage darauf abgebildete Objekte sowohl zu klassifizieren als auch zu lokalisieren (vgl. Redmon u. a., 2016). Dabei setzt es unter anderem auf Convolutional- und Pooling-Layer. Aufgrund der mannigfaltigen Anwendungsmöglichkeiten und der Beliebtheit des Objektdetektors wurde es über die Jahre von diversen Autoren weiterentwickelt (vgl. Redmon und Farhadi, 2016; Redmon und Farhadi, 2018; Bochkovskiy u. a., 2020; Jocher u. a., 2022).

Abbildung 2.11 fasst das Vorgehen des initialen Modells zusammen. Ein Bild besteht üblicherweise aus N x M Pixeln mit in der Regel jeweils einem Wert für die drei Farbkanäle rot, grün und blau. Diese N x M x 3 Matrix ist in der Bildverarbeitung unter dem Begriff Feature Map bekannt. YOLO nimmt eine solche Feature Map als Input entgegen und skaliert sie auf eine Größe von 448 x 448 x 3 (vgl. Redmon u. a., 2016). Anschließend wird das Bild in Zellstrukturen (SxS-Grid) untergliedert. Für jede Zelle wird eine Menge an Bounding Boxen prädiziert, in denen sich Objekte befinden

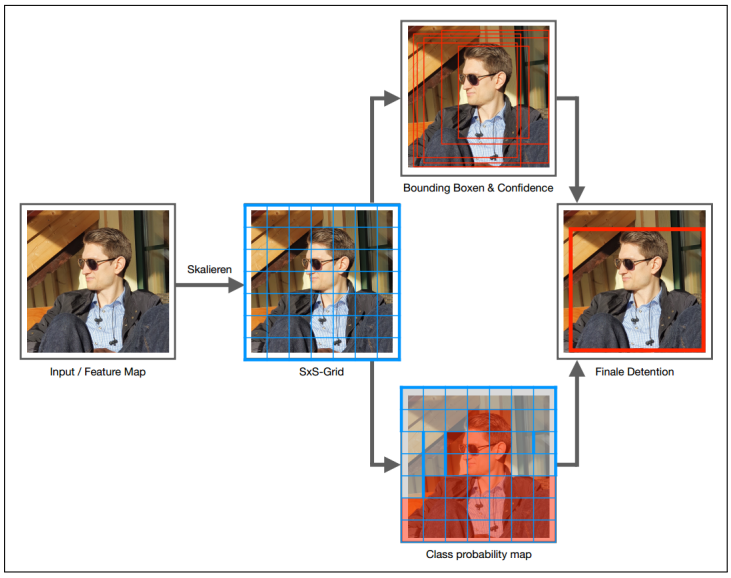


Abbildung 2.11: YOLOv1 Architektur. Eigene Darstellung in Anlehnung an (Redmon u. a., 2016, S. 780).

könnten. Zudem wird ein sogenannter Confidence Score berechnet, der angibt, mit welcher Wahrscheinlichkeit sich tatsächlich ein Objekt innerhalb der Bounding Box befindet. Weiterhin liefert das Modell eine Menge an Wahrscheinlichkeiten, mit denen sich Objekte eines bestimmten Typs (z.B. Person oder Hund) innerhalb der Gitterzelle befindet. Der Objekttyp mit der höchsten Wahrscheinlichkeit wird in einer Class Probability Map gespeichert. Letztlich kombiniert es die Ergebnisse vorausgegangener Schritte und führt eine sogenannte „non-maximal supression“ durch. Diese dient der Entfernung multipler Bounding Boxen für dasselbe Objekt.

Das Training und der Einsatz von Deep-Learning-Modellen mit komplexen und großen Architekturen (vielen Layern) erfordert leistungsfähige Hardware (vgl. Wang u. a., 2020). Dies gilt auch für die ersten YOLO-Versionen. Daher stellen viele Autoren eine abgespeckte Version ihrer Entwicklungen bereit. Diese enthalten weniger Layer, was mit einer Reduzierung des Rechenaufwands, aber auch einer Einbuße bei der Detektionsgenauigkeit einhergeht.

Gegenüber der ursprünglichen Variante stellt das von Jocher u.a. (2022) vorgestellte YOLOv5 (wobei v5 für die fünfte Version des Modells steht) eine deutliche Verbesserung in Bezug auf den Rechenaufwand und die Detektionsrate dar. Bis dato sind die Entwicklungen durch die Autoren nicht

in einer wissenschaftlichen Veröffentlichung publiziert. Ihr Modell besteht aus drei Hauptkomponenten:

- Backbone: Ein neuronales Netz, das Bildmerkmale in unterschiedlicher Granularität zusammenfasst und bildet. Die Autoren verwenden dafür das Cross Stage Partial Network (CSPNet) (vgl. Wang u. a., 2020). Dieses dient der Reduktion des benötigten Rechenaufwands bei gleichbleibender oder sogar besserer Detektionsgenauigkeit (vgl. Wang u. a., 2020).

- Neck: Ein weiteres neuronales Netz, das die zuvor identifizierten Bildmerkmale geschickt miteinander kombiniert und zur Vorhersage an den Head weiterleitet. Das sogenannte Path Aggregation Network (PANet) wurde zur Segmentierung von Bildern entwickelt (vgl. Liu u. a., 2018). Dabei werden Feature Maps in Regionen unterteilt, in denen sich Objekte befinden können (vgl. Liu u. a., 2018). Diese Fertigkeit nutzt das YOLOv5 Modell.

- Head: Ein neuronales Netz, welches auch in den vorherigen Versionen eingesetzt wurde. Es nimmt die Ergebnisse vorheriger Verarbeitungsschritte entgegen, prädiziert Bounding Boxen für Objekte, klassifiziert deren Inhalt und liefert einen Confidence Score.

Abbildung 2.12 visualisiert diese Hauptkomponenten und deren Relation zueinander. Wie bei seinen Vorgängern handelt es sich auch bei YOLOv5 um einen sogenannten Single-Stage Detektor (vgl. Bochkovskiy u. a., 2020). Ihren Namen erhalten diese Objekterkennungsmodelle von ihrer Fertigkeit, Bilder nur ein einziges Mal durch das Netzwerk zu schleusen und anschließend eine Lokalisation sowie Klassifikation der detektierten Objekte bereitzustellen. Als Input dient hierbei üblicherweise ein Bild (oder auch aneinander gereihte Bilder, beispielsweise in Form eines Videos). Das Ergebnis aller Verarbeitungsschritte stellen Koordinaten dar, welche die Position eines lokalisierten Objektes spezifizieren. Zusätzlich liefert das Modell den klassifizierten Typ und einen Grad der Zuversichtlichkeit, mit dem es sich um diesen Objekttyp handelt. Abbildung 2.13 verdeutlicht dies visuell. YOLOv5 steht als open-source Software auf der Plattform GitHub[4] zur Verfügung.

[4]Code einsehbar unter: https://github.com/ultralytics/yolov5/tree/v6.1

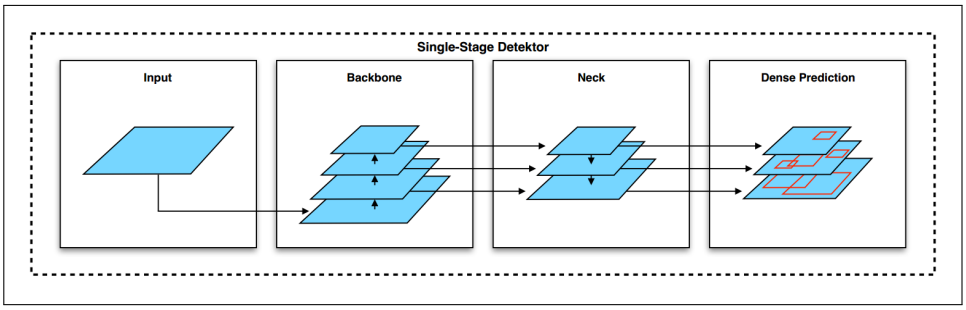


Abbildung 2.12: Single-Stage Architektur von YOLOv5. Eigene Darstellung in Anlehnung an (Bochkovskiy u. a., 2020, S. 2).

Das bereits vortrainierte Modell ist beispielsweise für die Detektion von Personen einsetzbar. Die Abbildung stammt aus dem privaten Archiv des Autors und zeigt eine Person. YOLOv5 detektiert diese mit einem Grad der Zuversichtlichkeit von 0.83 ($\hat{=}$ 83%). Eine rote Bounding-Box visualisiert die durch das Modell bestimmte Position der Person innerhalb des Bildes.

Besonders relevant im Kontext dieser Arbeit ist die Möglichkeit, den Objektdetektor auf die Erkennung weiterer Objekte trainieren zu können. Eine detaillierte Beschreibung dieses Prozesses findet sich beispielsweise bei GitHub[5]. Durch die Bereitstellung von eigenem Trainingsmaterial lässt sich YOLOv5 auf die Detektion von UML-Klassendiagrammkomponenten spezifizieren. Dies ermöglicht eine Untersuchung des visuellen Imports studentischer Abgaben respektive von Musterlösungen. Der Objektdetektor trägt daher in Teilen zur Beantwortung von Forschungsfrage 1.2 bei.

2.3 Studierendenzentriertes Lehren und Lernen

Die Literatur differenziert in diesem Kontext zwischen zwei unterschiedlichen Vorgehensweisen der Lehre: eine lehrendenzentrierte und eine studierendenzentrierte respektive lernorientierte (vgl. Kember, 1997; O'Neill und Mcmahon, 2005).

Bei ersterer stehen Lehrpersonen und die Vermittlung von Wissen im Mittelpunkt (vgl. Estes, 2004; vgl. R.M. Harden, 2000). Es handelt sich

[5]Einsehbar unter: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

Abbildung 2.13: Detektion und Lokalisation von Objekten in Bildern mithilfe des YOLOv5 Modells.

daher um ein inhaltsorientiertes Vorgehen.

Demgegenüber steht das studierendenzentrierte Lehren und Lernen. Es rückt als Wissensvermittler die Lernenden anstelle der Lehrenden in den Fokus. Dieser Ansatz bietet Studierenden die Möglichkeit, ihren Lernprozess individuell und aktiv zu gestalten (vgl. Estes, 2004; Hannafin u. a., 2014; Lee und Hannafin, 2016). Bei diesem lernorientierten Vorgehen verschieben sich die Bemühungen von Lehrpersonen von reinem Wissenstransfer (inhaltsorientiert) hin zur Begleitung sowie Unterstützung bei der Wissensaneignung. Lee und Hannafin (2016) empfehlen, dass Studierende bei studierendenzentriertem Lehren und Lernen ... (eigene Übersetzung in Anlehnung an (Lee und Hannafin, 2016, S. 707)):

- ... Eigenverantwortung über ihren Lernprozess entwickeln,
- ... persönliche und für sie sinnvolle Lernziele verfolgen,
- ... autonom lernen und dabei metakognitiv, prozedural, konzeptionell und strategisch durch eine Lehrperson unterstützt werden sollen.

Diese Gestaltungsform erfreut sich zunehmender Beliebtheit in der hochschulischen Lehre und kann zur intrinsischen Motivation von Studierenden beitragen, da sie statt einer passiven respektive konsumierenden eine aktive Rolle einnehmen müssen. Für Dozierende geht damit eine Anpassung ihrer Vorlesungs- und Übungskonzepte einher. Konkreter fasst es die Hochschulrektorenkonferenz im Jahre 2015 in einem Werk über Standards und Leitlinien für die Qualitätssicherung im Europäischen Hochschulraum zusammen. Für die Lehrpraxis bedeutet der Einsatz von studierendenzentriertem Lehren und Lernen:

- „die Diversität der Studierenden und ihrer Bedürfnisse zu respektieren und ihnen durch flexible Lernwege Rechnung zu tragen;
- wo es angebracht ist, unterschiedliche Vermittlungsweisen in Betracht zu ziehen und zu nutzen;
- unterschiedliche pädagogische Methoden flexibel einzusetzen;
- regelmäßige Evaluierungen und Anpassungen der Vermittlungsweisen und pädagogischen Methoden vorzusehen;
- die Studierenden zu selbstständigem Lernen zu ermutigen und ihnen als Lehrer gleichzeitig angemessene Orientierung und Unterstützung zu bieten;
- gegenseitigen Respekt in der Beziehung zwischen Lernenden und Lehrenden zu fördern;
- ein angemessenes Verfahren für den Umgang mit studentischen Beschwerden bereitzustellen.“ (Hochschulrektorenkonferenz, 2015, S. 20)

Das Erarbeiten eigener Lernstrategien ist ein zentraler Bestandteil dieses Ansatzes. Darunter sind Vorgehens- und Verhaltensweisen zu verstehen, die Studierende zur Erreichung eines Lernziels verfolgen (vgl. Weinstein und Mayer, 1983; vgl. Mandl und Friedrich, 1992). Sie definieren die Herangehensweise an Aufgabenstellungen und sind daher ein elementarer Bestandteil bei der Wissensaneignung.

2.4 Feedback

Feedback zählt zu den bedeutendsten Einflüssen, wenn es um das Erlernen neuer Fertigkeiten oder Inhalte geht (vgl. Hattie und Timperley, 2007; Lipnevich und Panadero, 2021). Feedback kann dabei als zyklischer Austausch von Informationen zwischen zwei oder mehr Entitäten (beispielsweise Personen oder Systemen) verstanden werden (vgl. Fengler, 2017). Eine merklich ausführlichere Definition von Feedback liefert Neudecker (2021):

> Feedback [...] ist eine Orientierung gebende Information, die von einer Person oder einem programmierten System als Reaktion auf ein Produkt/Abgabe/Handlung/Leistung einer Person (oder mehreren Personen) an Letztgenannte mit der Intention des Wissenszuwachs, des besseren Verstehens, der konstruktiven Weiterentwicklung, der Erhöhung an Qualität oder Motivation, der Entwicklung der Persönlichkeit oder der Professionalisierung zurückgemeldet wird. (Neudecker, 2021, S. 2)

Feedback ist folglich eine meist informelle Reaktion auf eine vorausgegangene Tätigkeit oder Leistung einer oder mehrerer handelnder Instanzen. Die rückgemeldeten Informationen müssen dabei direkten Bezug auf die vorausgegangene Handlung nehmen, um die Lücke zwischen dem zu schließen, was verstanden / ausgeführt wurde und dem was verstanden / ausgeführt werden sollte (vgl. Sadler, 1989; Hattie und Timperley, 2007).

In der Literatur werden die Begriffe Feedback und Assessment oftmals miteinander in Relation gebracht (vgl. Hattie und Timperley, 2007; Evans, 2013; Gauntlett, 2007). Sie sind jedoch nicht als Synonym zu verstehen. Assessment adressiert alle Prozesse oder Aktivitäten, die zur Beurteilung einer Tätigkeit oder Leistung dienen (vgl. Hattie und Timperley, 2007; Gauntlett, 2007). Feedback beinhaltet die darin transportieren Informationen.

In der hochschulischen Lehre stellt Assessment und damit einhergehendes Feedback einen zentralen Baustein in der Weiterentwicklung, Motivation und der Steigerung des Lernerfolgs von Studierenden dar (vgl. Hattie und Timperley, 2007). Laut Winne und Butler (1994) ist Feedback in diesem Kontext „[...] information with which a learner can confirm, add to, overwrite, tune, or restructure information in memory, whether that information is domain knowledge, meta-cognitive knowledge, beliefs about self

and tasks, or cognitive tactics and strategies“ (Winne und Butler, 1994, S. 5740; zitiert von: Hattie und Timperley, 2007, S. 82). Die kontinuierliche Bereitstellung dieser Informationen stellt einen essenziellen Bestandteil der hochschulischen Lehre dar und wurde umfassend erforscht (vgl. Hattie u. a., 2013; Hattie und Timperley, 2007). Trotz einer Vielzahl an Publikationen ist die Forschung in diesem Gebiet jedoch keineswegs abgeschlossen (vgl. Neudecker, 2021).

Da sich der Kontext dieser Arbeit mit der hochschulischen Lehre von UML-Klassendiagrammen in dem Fachgebiet Software Engineering befasst, kann im Rahmen dieser Arbeit eine Definition in Anlehnung an Fengler (2017), Neudecker (2021) sowie Winne und Butler (1994), wie folgt, aussehen:

> In der hochschulischen Lehre von UML-Klassendiagrammen im Studienfach Software Engineering wird Feedback als Orientierung gebende Information verstanden, die von einer dozierenden Person oder einem für diesen Lehrkontext ausgelegten System als Reaktion auf eine Tätigkeit, Handlung oder Abgabe von Studierenden rückgemeldet wird. Diese soll Studierende auf konstruktive Weise dabei unterstützen, bereits vorhandenes Wissen über UML-Klassendiagramme zu bestätigen und zu vertiefen oder sich neues Wissen und Strategien anzueignen. (in Anlehnung an: (Fengler, 2017, S. 16; Neudecker, 2021, S. 2; Winne und Butler, 1994, S. 5740))

2.4.1 Formen von Assessment

Die Beurteilung von Personen oder Systemen kann auf zwei verschiedene Weisen erfolgen. In der Literatur wird zwischen formativem und summativem Assessment differenziert (vgl. Gauntlett, 2007; Evans, 2013; Shute, 2008). Das Ziel von formativem Assessment ist die Einflussnahme auf das Handeln oder Denken einer oder mehrerer Instanzen, zur Weiterentwicklung oder Verbesserung. Die Besonderheit von formativem Feedback ist, dass es zeitnah nach einer Tätigkeit oder Leistung erfolgt und daher direkten Bezug auf den Handlungskontext nimmt (vgl. Evans, 2013; Shute, 2008). Zudem ist es meist nicht wertend (vgl. Kibble, 2017). Summatives

Assessment hingegen erfolgt nach Beendigung aller Prozesse und Aktivitäten, die der Weiterbildung oder Optimierung dienen und geht oftmals mit einer wertenden Leistungsbeurteilung einher (vgl. Kibble, 2017).

Die beschriebenen Formen von Assessment lassen sich anhand eines Beispiels aus der hochschulischen Lehre gut nachvollziehen. Im Rahmen eines Kurses werden oftmals wöchentliche Übungsstunden abgehalten, um die vorausgegangenen Kursinhalte zu repetieren und zu vertiefen. Eine nicht wertende bzw. benotende Beurteilung der Leistung von Studierenden erfolgt zeitnah und im selben Lehrkontext durch eine Lehrperson. Diese sollten jedoch nicht zu granular und häufig erfolgen, da sie ansonsten kontraproduktiv für den Lernerfolg der Studierenden wären (vgl. Hattie und Timperley, 2007; Evans, 2013). So ist es ausreichend, einzelne Aufgaben, aber nicht die individuellen Aktivitäten von Studierenden während der Bearbeitung dieser zu beurteilen. Die in diesem Kurs behandelten Lehrinhalte werden am Ende des Fachsemesters durch eine Klausur abgefragt. Mit Beendigung der Lehrperiode erfolgt eine summative und somit wertende Beurteilung der individuellen Wissensstände der Studierenden.

Wie an dem vorangegangenen Beispiel aus der Hochschullehre ersichtlich ist, treten beide Formen von Assessment oftmals in Kombination auf.

2.4.2 Kommunikation von Feedback in der hochschulischen Lehre

Eine konstruktive informelle Rückkopplung von Informationen erfolgt üblicherweise zwischen einem oder mehreren Adressaten und einem oder mehreren Adressierten. Tabelle 2.1 beinhaltet einen Ausschnitt der von Neudecker (2021) vorgestellten Pfade von Feedback in einem hochschulischen Kontext. Die Akteure können dabei laut Tabelle Dozierende, Studierende und / oder computergestützte Systeme sein. Es ist anzumerken, dass eine Person im Rahmen eines Selbstfeedbacks beide Rollen gleichzeitig innehaben kann (vgl. Neudecker, 2021). Am häufigsten erfolgt Rückmeldung von Dozierenden in Richtung der Studierenden, beispielsweise während den Übungsstunden eines Kurses.

Außerdem nennt Neudecker (2021) weitere Unterscheidungsmerkmale in der Kommunikation von Feedback. Diese werden im Folgenden in Anlehnung

an (Neudecker, 2021, S. 4f.) aufgelistet:

- Formale Struktur und Vollständigkeit.
- Analog oder digital.
- Synchron oder asynchron.
- In direktem (menschlichem) Kontakt oder durch Nutzung eines Mediums.

Durch die zunehmende Digitalisierung nimmt auch der Einsatz von computergestützten Systemen zu. Cavalcanti u. a. (2021) erstellten eine systematische Literaturübersicht zu automatisiertem Feedback in online Lernumgebungen. Die Ergebnisse zeigen, dass automatisiertes Feedback die Leistungen der Studierenden steigern können und dass der Großteil der analysierten Studien keinen Vorteil von herkömmlichem (persönlichem) gegenüber computergestütztem Feedback sieht (vgl. Cavalcanti u. a., 2021).

In der Kommunikation von Feedback ist jedoch nicht nur das Medium, sondern auch der transportierte Inhalt entscheidend. Hattie und Timperley (2007) zufolge beantwortet effektives Feedback drei grundlegende Fragen (Hattie und Timperley, 2007, S.86-90):

- Where am I going?
- How am I going?
- Where to next?

Die erste Frage (*Where am I going?*) beinhaltet in einem hochschulischen Kontext die Festlegung eines oder mehrerer zu erreichender Lernziele (vgl. Hattie und Timperley, 2007). Sofern das gewünschte Ergebnis für die Beteiligten nachvollziehbar ist, können zielgerichtete Handlungen geplant und durchgeführt werden (vgl. Bargh u. a., 2001; vgl. Hattie und Timperley, 2007), was zu einem verbesserten Lernergebnis führt. Die Orientierung gebende Information zu einer vorausgegangenen Tätigkeit, Handlung oder Abgabe (siehe Definition von Feedback in Abschnitt 2.4) wird nach Hattie und Timperley (2007) in der zweiten Frage (*How am I going?*) kommuniziert. Mit der Beantwortung dieser steht nun die Differenz zwischen dem gewünschten Ergebnis und der erbrachten Leistung fest. Frage drei (*Where*

Feedback Pfad	Beschreibung
D → ST	Von Dozierenden (D) zu Studierenden (ST).
ST → D	Von Studierenden (ST) zu Dozierenden (D).
ST → ST	Von Studierenden (ST) zu Studierenden (ST).
D → D	Von Dozierenden (D) zu Dozierenden (D).
SFB	Selbstfeedback (SFB) von Studierenden zu sich selbst.
SY → D	Von einem computergestützten System (SY) zu Dozierenden (D).
SY → ST	Von einem computergestützten System (SY) zu Studierenden (ST).

Tabelle 2.1: Feedback Pfade in Anlehnung an (Neudecker, 2021, S. 5).

to next?) erörtert, welche Schritte nun erfolgen können, um zukünftig ein besseres Ergebnis zu erzielen und somit die Lücke zwischen dem aktuellen und dem gewünschten Wissensstand zu schließen (vgl. Hattie und Timperley, 2007).

2.4.3 Typen von Feedback in einem hochschulischen Kontext

Aufgrund der hohen Aktivität in diesem Forschungsgebiet (vgl. Hattie und Timperley, 2007; Hattie u. a., 2013; Neudecker, 2021), sind mannigfaltige Formen von Feedback bekannt. Im Rahmen dieser Arbeit ist es besonders relevant zu erarbeiten, wie diese im Kontext der hochschulischen Lehre aussehen können.

Tabelle 2.2 beinhaltet eine Auflistung von zwölf Feedback Typen in Anlehnung an Shute (2008). Diese sind nach Komplexität in aufsteigender Reihenfolge sortiert und wurden in die deutsche Sprache übersetzt. Wie in Abschnitt 2.4.2 beschrieben, kann es in der hochschulischen Lehre diverse Adressaten und Adressierte geben. Shute (2008) geht von Dozierenden als Feedback gebende und von Studierenden als Feedback empfangende Personen aus. Je nach Lehrkontext kann es für erstere sinnvoll sein, auf unterschiedliche Typen von Feedback zurückzugreifen. Der Einsatz der simpelsten Form *Kein Feedback* ist jedoch selten von Mehrwert für die Studierenden, da Feedback nachweislich einen positiven Einfluss auf deren Lernerfolg hat

(vgl. Hattie und Timperley, 2007; Neudecker, 2021).

Typ	**Beschreibung**
Kein Feedback	Studierende erhalten keinerlei Feedback zu einer vorausgegangenen Abgabe, Tätigkeit oder Leistung.
Verifizierung	Studierende werden darüber informiert, ob ihre Antwort korrekt oder inkorrekt war (oder auch zu wie viel Prozent die Antwort korrekt war).
Richtige Antwort	Studierende erhalten die korrekte Antwort zu einer Aufgabe ohne weitere Informationen.
Erneuter Versuch	Studierende erhalten Informationen darüber, dass ihre Antwort nicht korrekt war und können eine (oder mehrere) erneute Antworten abgeben.
Fehlerkennzeichnung	Fehler einer studentischen Lösung werden markiert. Die Studierenden erhalten nicht die korrekte Antwort.
Elaboriert	Den Studierenden wird eine Erklärung bereitgestellt, warum ihre Antwort korrekt / inkorrekt war und bietet die Möglichkeit, Teile der Aufgabenbeschreibung zu überprüfen. Auch die Präsentation einer Musterlösung ist möglich.
Attribut Isolation	Elaboriertes Feedback, das zusätzlich zentrale Eigenschaften des Zielkonzepts präsentiert.
Themenbezug	Elaboriertes Feedback, das Studierende zusätzlich mit Informationen über das bearbeitete Thema versorgt.
Antwortbezug	Elaboriertes Feedback, das Studierenden zusätzlich vermittelt, warum ihre Antwort inkorrekt und eine andere korrekt ist.

Hinweise	Elaboriertes Feedback, das Studierenden zusätzlich Hinweise gibt, um sie der richtigen Antwort näher zu bringen. Die korrekte Antwort wird nicht präsentiert.
Falsche Vorstellungen	Elaboriertes Feedback, das Studierenden zusätzlich Informationen über ihre individuellen Fehler und ihrem falschen Verständnis liefert.
Informatives Lehren	Elaboriertes Feedback, das Studierenden zusätzlich eine Verifikation, Fehlerkennzeichnung und strategische Hinweise liefert, wie sie weiter Verfahren können. Üblicherweise wird die korrekte Antwort nicht präsentiert.

Tabelle 2.2: Typen von Feedback. Eigene Darstellung in Anlehnung an (Shute, 2008, S. 10).

2.4.4 Ein Feedback Modell im hochschulischen Kontext

Eines der gängigsten und anerkanntesten (vgl. Lipnevich und Panadero, 2021) Feedback Modelle wurde von Hattie und Timperley (2007) vorgestellt. Dieses wurde in Abbildung 2.14 visualisiert. Die Reduktion der Differenz zwischen einem aktuellen und einem gewünschten Wissensstand steht für Hattie und Timperley (2007) an oberster Stelle, wie aus Abbildung 2.4 zu entnehmen ist. Diese kann durch Studierende (beispielsweise durch Anwendung neuer Lernstrategien), aber auch durch Dozierende (beispielsweise durch Bereitstellung angemessener, aber herausfordernder Aufgaben) erfolgen (vgl. Hattie und Timperley, 2007). Um diesen Prozess effektiv zu gestalten, müssen die in Abschnitt 2.4.2 vorgestellten Fragen beantwortet werden. Inhaltlich können sich die Antworten dieser Fragen auf vier unterschiedliche Level beziehen. Das Task Level bietet Aufschluss darüber, wie erfolgreich eine bereitgestellte Aufgabe bearbeitet wurde (vgl. Hattie und Timperley, 2007). Das Prozess Level nimmt Bezug auf die verwendete Vorgehensweise, die zur Erfüllung der Aufgabe verwendet wurde (vgl. Hattie und Timperley, 2007). Das Selbstregulierungs Level zeigt auf, wie Studierende ihre Handlungen zur Erreichung eines Ziels planen, überwachen und

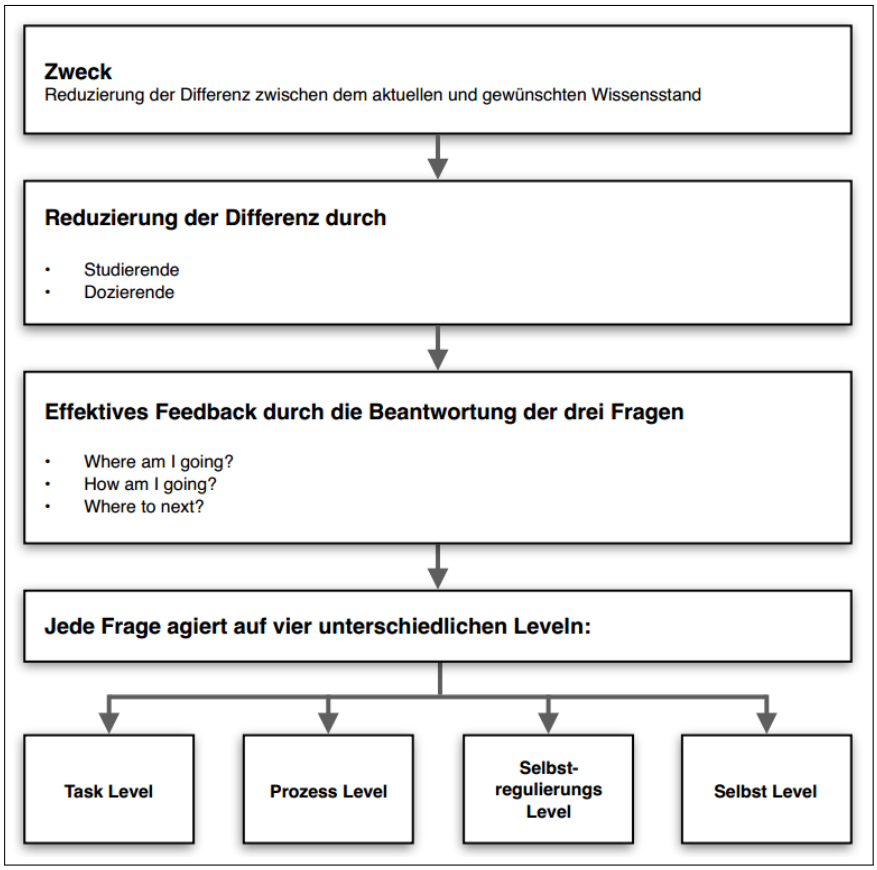


Abbildung 2.14: Feedback Modell zur Verbesserung des Lernerfolgs nach (Hattie und Timperley, 2007, S. 87).

kontrollieren können (vgl. Hattie und Timperley, 2007). Das Selbst Level stellt Feedback zur Person bereit (beispielsweise: „Du bist schlau!“) und ist das am wenigsten effektive, da es keinerlei inhaltlichen Bezug nimmt (vgl. Hattie und Timperley, 2007).

Das Feedback Modell wird im Folgenden für die Bereitstellung einer Orientierung gebenden Rückmeldung für Studierende herangezogen. Es soll als Ausgangsbasis zur Kommunikation bestehender Differenzen einer studentischen Lösung zu einer Musterlösung sowie Möglichkeiten zur künftigen Leistungssteigerung dienen. Folglich stellt es einen elementaren Bestandteil für die Beantwortung von Forschungsfrage 1.3 dar.

KAPITEL 3

Anforderungserhebung

Nach der Erörterung der theoretischen Grundlagen erfolgt in diesem Kapitel eine Anforderungserhebung mit den jeweiligen Zielgruppen an das zu konzipierende Gesamtsystem. In einem ersten Schritt wird, im Rahmen des DSR-Forschungsdesigns nach Hevner u. a. (2004), die im Fachbereich Software Engineering bestehende Wissensbasis untersucht. Konkret soll eine Systematische Literaturübersicht (SLR) zu den bestehenden, in der Literatur beschriebenen Softwarelösungen für die Unterstützung der hochschulischen Lehre von UML-Diagrammen erfolgen. Um einen vollständigeren Überblick über die aktuelle Wissensbasis zu erlangen, liegt der Fokus nicht ausschließlich auf Klassendiagrammen. Weiterhin ist die Erhebung von Anforderungen der Hauptakteure (Dozierende und Studierende) ein entscheidender Aspekt in diesem Kapitel. Diese erfolgt mithilfe leitfadengestützter Experteninterviews. Die daraus resultierende Gesamtübersicht relevanter Funktionalitäten kann als Entwicklungsgrundlage bei den darauffolgenden Konzeptionen dienen.

3.1 Systematische Literaturübersicht zur toolgestützten Lehre von Software Engineering

Eine SLR, die auch unter dem englischen Begriff „Systematic Literature Review“ bekannt ist, dient der umfassenden Analyse der bestehenden Wissensbasis eines spezifischen Themenschwerpunkts. Dabei handelt es sich um eine anerkannte, strukturierte Vorgehensweise für die Identifikation und Konsolidierung von Primärliteratur (vgl. Kitchenham, 2004; vgl. Kitchenham und Charters, 2007). Die resultierenden Ergebnisse können als Entwicklungsgrundlage für weitere Forschungsansätze herangezogen werden und helfen zu entwickelnde Artefakte und Theorien von bereits existierenden zu differenzieren.

Die Wissensbasis stellt unterschiedliche Methodiken und Werkzeuge bereit, welche eine strukturierte Durchführung einer SLR ermöglichen und durch die Forschungsgemeinschaft vielfach erprobt sind. Eine in den Forschungsschwerpunkten Informatik und Software Engineering häufig verwendete Vorgehensweise wurde von Kitchenham (2004) sowie von Kitchenham und Charters (2007) vorgestellt. Die Autoren präsentieren drei Phasen, die eine SLR durchlaufen muss (in Anlehnung an (vgl. Kitchenham, 2004; vgl. Kitchenham und Charters, 2007), übersetzt durch den Autor):

1. Planung der SLR.
2. Durchführung der SLR.
3. Präsentation der SLR.

Die erste Phase beinhaltet die Planung der SLR. Kitchenham und Charters (2007) zufolge ist hierfür ein ausführliches Protokoll zu erstellen, das die folgenden zentralen Punkte enthält (übersetzt durch den Autor):

a. Definition der Forschungsfrage(n).
b. Zu durchsuchende elektronische Datenbanken.
c. Definition von Einschluss- und Ausschlusskriterien.
d. Auswahl der Primärliteratur anhand komplexer Recherche mithilfe zusammengesetzter Suchbegriffe.

e. Datenextraktion und -synthese.

In Phase zwei erfolgt die Durchführung der SLR. Diese orientiert sich anhand der im Protokoll erarbeiteten Kriterien. Schlussendlich kann das SLR in der dritten Phase präsentiert werden.

Nach den Vorgaben von Kitchenham (2004) sowie von Kitchenham und Charters (2007) soll eine Untersuchung bestehender Literatur erfolgen, die softwarebasierte Anwendungen für die hochschulische Lehre von UML-Diagrammen beschreibt. Darin enthaltene Erkenntnisse und Funktionalitäten lassen sich anschließend in Anforderungen überführen.

Ein von Huber und Hagel (2022b) veröffentlichtes SLR befasst sich mit dieser Thematik. Es beinhaltet den festgelegten Kriterien entsprechende Fachliteratur, die zwischen dem 1. Januar 2010 und dem 31. Dezember 2021 publiziert wurde. Das Verfahren und die Ergebnisse finden sich in den folgenden Unterabschnitten. Der Aufbau orientiert sich an den bereits vorgestellten Phasen einer SLR. Zusätzlich erfolgt eine Weiterführung für den Zeitraum zwischen dem 1. Januar 2022 und dem 15. Mai 2023. Weiterhin wird eine Suche über die elektronische Datenbank Google Scholar durchgeführt, um weitere, nicht in den von Huber und Hagel (2022b) angeführten Datenbanken enthaltene Literatur zu detektieren (siehe Unterabschnitt 3.1.2). Obgleich in dieser Arbeit die UML-Klassendiagramme im Fokus stehen, wird die Wissensbasis nach Anwendungen für alle UML-Diagramme untersucht, da vorgestellte Konzepte auf unterschiedliche Diagrammtypen übertragbar sein können.

3.1.1 Planung der SLR

Die für ein SLR Protokoll relevanten Punkte nach Kitchenham und Charters (2007) wurden von Huber und Hagel (2022b) aufgegriffen und beschrieben. Sie sollen auch in diesem Abschnitt als Grundlage für das Protokoll dienen. Die Inhalte der folgenden Unterabschnitte sind in Anlehnung an Huber und Hagel (2022b) entstanden und wurden ggf. durch den Autor in die deutsche Sprache übersetzt.

a. Definition der Forschungsfrage(n)

Die Definition von Forschungsfragen erfolgt in einem ersten Schritt, da sie die Ziele der SLR abstecken und als Grundvoraussetzung für die Bearbeitung der weiteren Protokollpunkte herangezogen werden (vgl. Kitchenham, 2004; vgl. Kitchenham und Charters, 2007). Huber und Hagel (2022b) definieren drei Forschungsfragen:

- **SLR-F1:** Welche Softwareanwendungen werden zur Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering eingesetzt?
- **SLR-F2:** Welche fundamentalen Differenzen bestehen zwischen diesen Softwareanwendungen?
- **SLR-F3:** Welche Defizite sollen durch den Einsatz dieser Softwareanwendungen eliminiert werden?

b. Zu durchsuchende elektronische Datenbanken

Die Auswahl der zu durchsuchenden, elektronischen Datenbanken ist entscheidend, um aussagekräftige Literatur zu finden, anhand derer sich die Forschungsfragen beantworten lassen. Für das Forschungsgebiet Software Engineering besonders relevante Datenbanken sind (vgl. Chen u. a., 2010; vgl. Kitchenham und Charters, 2007; vgl. Brereton u. a., 2007; vgl. Huber und Hagel, 2022b):

- IEEExplore (https://ieeexplore.ieee.org).
- ACM Digital Library (https://dl.acm.org/).
- ScienceDirect (https://www.sciencedirect.com/).
- Wiley Online Library (https://onlinelibrary.wiley.com/).
- SpringerLink (https://link.springer.com/).

Da Huber und Hagel (2022b) keinen Zugriff auf die Scopus[6] Datenbank hatten, wurde zusätzlich eine Google Scholar[7] Suche durchgeführt. Das Vorgehen dieser Arbeit orientiert sich an diesen Angaben.

[6] https://www.scopus.com
[7] https://scholar.google.com/

Identifikator	Einschlusskriterium
SLR-I1	Wissenschaftliche Artikel, Konferenzbeiträge oder vergleichbare Publikationen, die Peer-Reviewed sind.
SLR-I2	Publikationen müssen in deutscher oder englischer Sprache verfügbar sein.
SLR-I3	Publikationen, die zwischen dem 1. Januar 2010 und dem 31. Dezember 2021 veröffentlicht wurden.
SLR-I4	Die Publikation muss eine Softwareanwendung beschreiben, die zur Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering verwendet wird.

Tabelle 3.1: Einschlusskriterien der SLR. Eigene Darstellung in Anlehnung an (Huber und Hagel, 2022b, S. 1405).

c. Definition von Einschluss- und Ausschlusskriterien

Die Einschluss- und Ausschlusskriterien geben an, unter welchen Bedingungen gefundene Literatur in die Literaturübersicht aufgenommen oder ausgegrenzt wird. Die in diesem Kontext definierten Kriterien finden sich jeweils in Tabelle 3.1 und 3.2.

Den Angaben von Rowley und Slack (2004) zufolge, sollen wissenschaftliche Publikationen (beispielsweise aus wissenschaftlichen Journalen oder Konferenzbänden) die Grundlage für eine SLR darstellen, da diese Peer-Reviewed und somit vor der Veröffentlichung einer Prüfung durch die Fachgemeinschaft unterzogen sind (vgl. Rowley und Slack, 2004). Dadurch lässt sich sicherstellen, dass die Beantwortung der Forschungsfragen anhand qualitativ geprüfter Literatur erfolgt. Das Einschlusskriterium SLR-I1 legt dies für die folgende SLR fest. Weiterhin muss eine Publikation in deutscher oder englischer Sprache verfügbar sein. Andernfalls können durch den Autor keine stichhaltigen Aussagen über deren Inhalt getroffen werden. Diese Einschränkung ist durch SLR-I2 festgehalten. Auch eine zeitliche Eingrenzung findet durch SLR-I3 statt. Wie von Huber und Hagel (2022b) argumentiert, lassen sich dadurch vergleichsweise aktuelle Softwareanwendungen in der Li-

Identifikator	Ausschlusskriterium
SLR-A1	Sekundärliteratur.
SLR-A2	Work-In-Progress Publikationen, bei denen noch kein konkretes System oder Konzept identifizierbar ist.
SLR-A3	Lehrmethoden, die nicht softwaregestützt sind.
SLR-A4	Publikationen, die eine Softwareanwendung zur Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering nennen, deren Funktionsweise und Einsatzspektrum jedoch nicht beschreiben.
SLR-A5	Theoretische Frameworks, Konzeptionen, Ideen, Visionen die bis dato nicht realisiert wurden.

Tabelle 3.2: Ausschlusskriterien der SLR. Eigene Darstellung in Anlehnung an (Huber und Hagel, 2022b, S. 1405).

teratur finden, die auch heute noch einsetzbar wären. Die Akzeptanz einer solchen bei Studierenden ist von zentraler Bedeutung für deren Erfolg. Eine benutzerfreundliche Bedienoberfläche im modernen Design ist dafür eine Grundvoraussetzung. Die zeitliche Eingrenzung ist daher bewusst gewählt. Das zuletzt gelistete Einschlusskriterium SLR-I4 hält fest, dass eine Publikation eine Softwareanwendung beschreiben muss, die für die Lehre von UML-Diagrammen für die hochschulische Lehre von Software Engineeering einsetzbar ist. Andernfalls ist keine Relevanz im Kontext der SLR gegeben.

Neben den vier Einschlusskriterien sind auch fünf Ausschlusskriterien definiert. In SLR-A1 ist festgehalten, dass keine Sekundärliteratur einbezogen wird. Außerdem sind durch SLR-A2 Work-In-Progress Publikationen ausgeschlossen, sofern diese keine konkrete Softwareanwendung oder kein konkretes, softwaregestütztes Konzept präsentieren. Auch Lehrmethoden, die nicht softwaregestützt sind, werden nicht inkludiert (siehe SLR-A3). Weiterhin sind Publikationen ausgeschlossen, die eine solche Softwareanwendung nennen, deren zugrundeliegende Konzeption und Funktionsweise jedoch nicht beschreiben (siehe SLR-A4). Auch theoretische Frameworks, Konzeptionen, Ideen oder Visionen, die bis dato nicht realisiert sind finden wie in SLR-A5 beschrieben keinen Einzug in die SLR.

d. Auswahl der Primärliteratur anhand komplexer Recherche mithilfe zusammengesetzter Suchbegriffe

Die Suche innerhalb der vorgestellten, elektronischen Datenbaken erfolgt mittels zusammengesetzter Suchbegriffe (vgl. Kitchenham, 2004; vgl. Kitchenham und Charters, 2007). Aufgrund der Heterogenität dieser Datenbanken ist jedoch die Verwendung einer identischen Suchanfrage nicht überall möglich (vgl. Brereton u. a., 2007; zitiert von: Huber und Hagel, 2022b; S. 1405). Adaptiert auf die jeweiligen Eingabemöglichkeiten bleibt er jedoch in seiner Essenz erhalten (in Anlehnung an (Huber und Hagel, 2022b; S. 1405)):

TITLE-ABS-KEY *((„tool“* ***OR*** *„system“* ***OR*** *„approach“* ***OR*** *CASE)* ***AND*** *„software engineering“* ***AND*** *„education“* ***AND*** *(„UML“* ***OR*** *„diagram“* ***OR*** *„modelling“))*
„filter“: {Publication date: (01/01/2010 TO 12/31/2021)}

Eine Kombination gewünschter Suchbegriffe ist durch eine Verknüpfung mithilfe der Signalworte ***AND*** respektive ***OR*** möglich. Ersteres bestimmt, dass die gewählten Worte in Verbindung miteinander auftreten müssen, wohingegen Letzteres angibt, dass einer der Begriffe aus einer vorgegebenen Menge enthalten sein muss.

e. Datenextraktion und -synthese

Folgende Informationen werden für die Gesamtheit der inkludierten Veröffentlichungen erfasst:

- Die Anzahl der insgesamt selektierten Publikationen.
- Eine vollständige Liste selektierter Publikationen.

Individuell für jede Veröffentlichung werden die folgenden Informationen erfasst:

- Titel.
- Link.

- Datum der Suche.
- Name der entwickelten Softwareanwendung (falls vorhanden).
- Funktionalitäten der Softwareanwendung.
- Defizite, die durch den Einsatz dieser Softwareanwendungen angegangen werden sollen.

3.1.2 Durchführung der SLR

Nach Erstellung des Protokolls und dem damit einhergehenden Abschluss der ersten SLR Phase, erfolgt die Durchführung. Diese ist durch den Autor, in Anlehnung an (Huber und Hagel, 2022b, S. 1405f.) und (Huber und Hagel, 2021, S. 1620) in vier Schritte untergliedert und in Abbildung 3.1 einsehbar.

Der erste Schritt beinhaltet eine initiale Suche in den elektronischen Datenbanken unter Verwendung der vorgestellten, zusammengesetzten Suchbegriffe. Das Ergebnis stellen 171 potenziell relevante Publikationen dar. Daraufhin wurden die Einschluss- und Ausschlusskriterien im zweiten Schritt auf den Titel, die Kurzzusammenfassung, die Schlüsselwörter sowie das Fazit angewendet. Daraus resultierte eine Anzahl von 44 Publikationen. Anschließend wurden diese vollständig gelesen und anhand der Einschluss- und Ausschlusskriterien erneut bewertet. Die 29 in Schritt drei selektierten Publikationen finden Einzug in die SLR. Im vierten Schritt erfolgte außerdem die Durchführung eines Schneeballverfahrens zur Identifikation weiterer relevanter Literatur. In einem rückwärts gerichteten Verfahren werden dabei die Literaturhinweise bereits selektierter Publikationen auf weitere relevante Veröffentlichungen untersucht (vgl. Budgen u. a., 2008; vgl. Baur und Blasius, 2019). Zusätzlich ist in einem vorwärts gerichteten Verfahren zu überprüfen, welche Veröffentlichungen die selektierten Publikationen zitieren (vgl. Budgen u. a., 2008; vgl. Baur und Blasius, 2019). Das Endergebnis stellen 41 Publikationen dar.

3.1.3 Präsentation der Ergebnisse der SLR

SLR-F1: Welche Softwareanwendungen werden zur Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering eingesetzt?

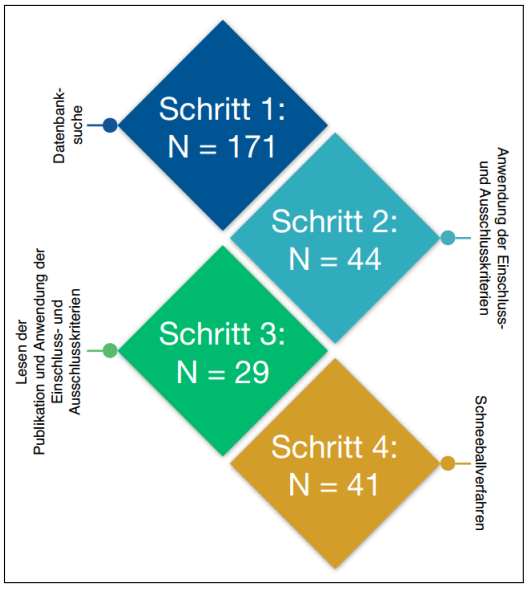


Abbildung 3.1: Vier Schritte zur Durchführung der SLR. Eigene Darstellung in Anlehnung an (Huber und Hagel, 2022b, S. 1405f.; Huber und Hagel, 2021, S. 1620).

Die Präsentation der Ergebnisse ist in zwei Schritte untergliedert. Zuerst sind die von Huber und Hagel (2022b) beschriebenen Softwareanwendungen angeführt sowie deren Funktionalitäten erläutert. Simultan erfolgt eine Beschreibung identifizierter Literatur, welche durch die Weiterführung der SLR in dieser Arbeit zusätzlich identifiziert werden konnte. Eine Sortierung der Publikationen erfolgt thematisch. Vergleichbare Ansätze sind möglichst nacheinander angeführt. Darüber hinaus sind sie bei der Beantwortung von Forschungsfrage SLR-F2 in induktiv gebildete Gruppierungen unterteilt. Eine solche Separierung ist jedoch erst vorzunehmen, wenn alle Inhalte der inkludierten Publikationen bekannt sind und muss daher zu einem späteren Zeitpunkt erfolgen.

Softwareanwendungen für die Lehre von UML-Diagrammen zur Unterstützung der hochschulischen Lehre von Software Engineering (in Anlehnung an (Huber und Hagel, 2022b, S. 1406f.), übersetzt durch den Autor):

- D. Stikkolorum, Ho-Quang u. a. (2015) präsentieren einen online UML-Editor für die Erstellung von Diagrammen (vgl. Stikkolorum u. a., 2015). Die Softwareanwendung loggt und visualisiert die Aktivitäten von Studierenden. Dozierende können diese Ergebnisse einsehen, De-

fizite respektive häufige Fehler identifizieren und durch entsprechende Maßnahmen reduzieren.

- *JavalinaCode* ist ein interaktiver online Code-Editor (vgl. Yang u. a., 2015) und bietet Studierenden die Möglichkeit, Programmcode zu einer Aufgabe zu entwickeln, an ein Backend zu senden und sich ein zugehöriges UML-Klassendiagramm anzeigen zu lassen. Eine Ausführung des Codes ist ebenfalls über die Oberfläche möglich. Zur Laufzeit visualisiert diese, welche Werte gesetzt und verändert werden.

- *JaguarCode* stellt einen online Code-Editor bereit, der die statische Struktur und das dynamische Verhalten von Java Code zur Laufzeit aufzeigt (vgl. Yang u. a., 2017). Es verfolgt einen ähnlichen Ansatz wie das zuvor beschriebene *JavalinaCode*, indem er studentischen Programmcode in ein UML-Klassendiagramm umwandelt, anzeigt und darüber hinaus visualisiert, wie sich die Attributwerte von Objekten zur Laufzeit des Codes verändern.

- *Umple* ist ein online Editor (auch als Konsolenanwendung oder für die Entwicklungsumgebung Eclipse[8] verfügbar), der Studierenden die Erstellung von Programmcode in eigener Syntax oder die Modellierung eines UML-Diagramms erlaubt (vgl. Lethbridge u. a., 2011). Entwickelter Programmcode kann in ein UML-Diagramm und vice versa umgewandelt werden. Studierende erkennen somit den direkten Zusammenhang zwischen diesen beiden Komponenten. Zudem kann der Programmcode in unterschiedliche Programmiersprachen (beispielsweise Java) übersetzt werden.

- Ein ähnlicher Ansatz wird von Krusche u. a. (2020) verfolgt. Die Autoren zeigen einen online Code-Editor (vgl. Krusche u. a., 2020). Er bietet Studierenden die Möglichkeit, Programmcode hinzuzufügen und ein angezeigtes UML-Diagramm zu implementieren. Ist die Entwicklung einer Komponente erfolgt, wechselt diese im Diagramm auf die Farbe grün. Zusätzlich werden Beschreibungen und Hinweise unterhalb des Diagramms angezeigt. Studierende erhalten hier direktes Feedback zu ihrem entwickelten Programmcode.

[8] https://www.eclipse.org/ide/

- *VisAr3D* konvertiert 2D UML-Diagramme in 3D Visualisierungen und erlaubt eine Exploration (großer) Modellierungen aus unterschiedlichen Perspektiven, Abstraktionsebenen etc. (vgl. Rodrigues u. a., 2016).

- *Ariadne* ermöglicht es Studierenden UML-Diagramme in 2D, aber auch in Augmented Reality Ansicht anzusehen und zu bearbeiten (vgl. Reuter u. a., 2019). Die Autoren verwenden dafür eine Microsoft HoloLens[9] Brille.

- *UMLint* ist eine online Anwendung, die Studierenden Feedback zu erstellten UML-Diagrammen bereitstellt (vgl. Hasker und Rowe, 2011). Die Anwendung analysiert Modelldateien, die mit der Modellierungsumgebung IBM Rational Rose[10] erstellt wurden und untersucht diese auf wiederkehrende Fehler. Die Anwendung kann nicht verifizieren, ob ein anhand von Anforderungsbeschreibungen erstelltes Diagramm korrekt und vollständig ist.

- Sedrakyan und Snoeck (2012) präsentieren einen simplifizierten Editor, mit dessen Hilfe Studierende Diagramme erstellen können (vgl. Sedrakyan und Snoeck, 2012). Sie erhalten eine textuelle Anforderungsbeschreibung und müssen ein entsprechendes Diagramm entwickeln. Die Softwareanwendung stellt ihnen Tipps und einfache Diagnosen bereit. Lehrpersonen können die Diagnosen einsehen sowie gezielt auf häufige Fehler reagieren.

- Stikkolorum u. a. (2019) stellen eine experimentelle Studie vor, in der Studierende UML-Diagramme erstellen mussten, die anschließend durch Experten bewertet wurden (vgl. Stikkolorum u. a., 2019). Mit diesen Daten trainierten die Autoren unterschiedliche Machine Learning Modelle, mit dem Ziel einer automatisierten Bewertung studentischer UML-Diagramme. Nach eigenen Angaben der Autoren sind die Modelle bis dato jedoch nicht ausgereift genug, um Einzug in den Lehrbetrieb zu finden.

[9] https://www.microsoft.com/en-us/hololens

[10] https://www.ibm.com/support/pages/node/306477?mhsrc=ibmsearch_a&mhq=Rational%20rose

- Boubekeur u. a. (2020) stellen einen ähnlichen, durch Heuristik und Machine Learning gestützten Softwareansatz vor (vgl. Boubekeur u. a., 2020). Studierende modellieren dabei ein Diagramm mithilfe der bereits vorgestellten Softwareanwendung *Umple*. Anschließend benotet das von Boubekeur u. a. entwickelte System mit einer Bewertungsskala von A (Bestnote) bis F (schlechteste Note).

- *CaMeLOT* ist eine auf Studierende zentrierte, personalisierte Softwareanwendung, die mithilfe von Domänenwissen und einer Liste häufiger Fehler UML-Diagramme bewertet und Feedback bereitstellt (vgl. Bogdanova, 2019). Studierende erhalten auf ihren individuellen Wissensstand angepasste Aufgaben. Entdeckt die Softwareanwendung einen Fehler in der Modellierung, stellt sie eine weitere Aufgaben mit demselben Schwierigkeitsgrad bereit. Falls keine Fehler detektiert werden können, liefert sie anspruchsvollere Aufgaben. Die Erstellung der Aufgabentexte erfolgte manuell.

- Ichinohe u. a. (2019) publizierten eine Softwarelösung, die Studierenden die Modellierung von UML-Klassendiagrammen ermöglicht und die Ergebnisse bewertet (vgl. Ichinohe u. a., 2019). Die Anwendung loggt durchgeführte Aktivitäten und vergleicht die darin enthaltenen Informationen mit einer Musterlösung, die zuvor hinterlegt wurde. Lehrpersonen prüfen Namensgebungen mit synonymen Bedeutungen und können diese ggf. in der Softwareanwendung als korrekte Antwort hinterlegen. Mithilfe einer Formel bewertet sie die studentischen UML-Klassendiagramme, wobei 1.0 das beste und 0.0 das schlechteste Ergebnis darstellt.

- *UML Diagram Learning (UDL)* ist eine Softwareanwendung, die UML-Use-Case-Diagramme und UML-Klassendiagramme anhand eines vordefinierten Regelwerks bewertet und Feedback generiert (Ali u. a., 2018). Dozierende stellen eine Aufgabe bereit, die von Studierenden bearbeitet wird. Ein Vergleich studentischer Lösungen mit einer Musterlösung findet nicht statt.

- *UMLtest* ist eine Softwareanwendung, die von Dozierenden erstellte UML-Klassendiagramme als Ausgangspunkt verwendet, diese in Pro-

grammcode umwandelt und automatisiert zugehörige Unit-Tests erstellt (vgl. Herout und Brada, 2016). Studentische Lösungen werden ebenfalls in Programmcode transformiert. Die vorhandenen Unit-Tests lassen sich anschließend für den studentischen Programmcode ausführen. Nach dem ersten detektierten Fehler stoppt die Anwendung und stellt eine Rückmeldung bereit. Dozierende erhalten eine Liste der Fehler ihrer Studierenden. Für die Modellierung der Diagramme verwenden die Autoren die Modellierungsumgebung UMLet[11].

- *ModBud* ist eine auf künstlicher Intelligenz basierende Softwareanwendung, die Studierenden Hilfestellung bei der Modellierung sowie Feedback zu erstellten UML-Diagrammen bereitstellt (vgl. Saini u. a., 2019). Über einen Dozierenden-Modus können Lehrpersonen Aufgabenstellungen und Musterlösungen hinterlegen. Zusätzlich besteht die Möglichkeit einem Chatbot über natürliche Sprache Hintergrundinformationen zu einer Aufgabe mitzuteilen. Die Softwareanwendung lernt die Zusammenhänge zwischen einem Aufgabentext und der zugehörigen Musterlösung. Unter Verwendung des vorhandenen Wissens und mithilfe des Chatbots gibt sie Studierenden Hilfestellung sowie Feedback.

- *PREViA* ist eine Softwareanwendung, die Unterschiede in verschiedenen Evolutionsstufen einer Softwarearchitektur / eines Softwaremodells detektieren kann (vgl. Schots u. a., 2010). Studierende erhalten eine Aufgabenstellung und erstellen ein zugehöriges UML-Diagramm. Dieses wird mit einer durch die Dozierenden bereitgestellten Musterlösung verglichen.

- Bain u. a. (2019) stellen eine Softwareanwendung zur automatischen Benotung von UML-Klassendiagrammen vor (vgl. Bian u. a., 2019). Die Modelldateien einer studentischen und einer Musterlösung werden miteinander verglichen und deren Differenzen detektiert.

- Wei u. a. (2016) präsentieren eine Softwareanwendung, die studentische UML-Diagramme in konzeptuelle Graphen umwandelt und diese

[11] http://www.umlet.com/

mit von Dozierenden definierten Informationen (Musterlösung) vergleicht (vgl. Wei u. a., 2016). Die Studierenden erhalten Feedback, in dem die Anwendung iterativ fehlende Elemente in das studentische Diagramm einfügt und diese markiert.

- Cai u. a. (2011) publizierten eine Softwareanwendung, die eine studentische und eine Musterlösung in Matrizen transformiert, welche anschließend miteinander verglichen werden konnten (vgl. Cai u. a., 2011). Attribute und Methoden von Klassen werden dabei jedoch nicht berücksichtigt.

- *UMLGrader* ist eine online Softwareanwendung, die studentische Diagramme mit einer Musterlösung vergleicht und Feedback generiert (vgl. Hasker, 2011). Studierende können ihre Diagramme anhand dieser Rückmeldung überarbeiten und erneut abgeben. Erkennt die Softwareanwendung keine weiteren Fehler, leitet sie das Diagramm an Dozierende weiter, die eine finale Bewertung vornehmen.

- Reischmann und Kuchen (2016) stellen eine E-Assessment Anwendung vor, die in die Lernplattform Moodle[12] integrierbar ist (vgl. Reischmann und Kuchen, 2016). Sie soll Studierende beim Erlernen von Design Patterns bei UML-Klassendiagrammen unterstützen. Dafür werden studentische Lösungen in Graphen umgewandelt und mit einer oder mehreren möglichen Musterlösungen (bereitgestellt durch Dozierende) verglichen. Das Feedback beinhaltet detektierte Differenzen und gibt Hinweise, falls das falsche Design Pattern verwendet wurde.

- *DUDE* ist eine durch Deep-Learning-Methoden gestützte Softwareanwendung, die studentische Diagramme visuell importiert, deren Informationen extrahiert und mit einer Musterlösung vergleicht (vgl. Huber und Hagel, 2020). Der visuelle Import ermöglicht es Studierenden, eine Modellierungsumgebung ihrer Wahl zu verwenden oder diese gar händisch auf einem Blatt Papier zu erstellen. Eine Auswertung handschriftlicher Diagramme ist laut der Autoren prinzipiell mit diesem Ansatz möglich, jedoch nicht explizit implementiert.

[12] https://moodle.org/

- Modi u. a. (2021) beschreiben eine Softwareanwendung, die Modellierungsdateien einlesen und miteinander vergleichen kann (vgl. Modi u. a., 2021). Studentische UML-Diagramme können so mit Musterlösungen verglichen und deren Differenz als Feedback kommuniziert werden. Durch einen Bewertungsalgorithmus erhalten die Studierenden außerdem eine Benotung.

- *SD4ED* ist eine Softwareanwendung, die Studierenden Feedback bei der Entwicklung von UML-Sequenzdiagrammen bereitstellt (vgl. Alhazmi u. a., 2021). Als Ausgangsbasis verwendet sie ein UML-Klassendiagramm. Die Softwareanwendung prüft, ob alle in dem Klassendiagramm enthaltenen Komponenten auch in dem Sequenzdiagramm vorhanden sind. Zusätzlich erkennt sie ungültige Aktivitäten und liefert Studierenden Hinweise.

- Soler u. a. (2010) präsentieren eine Softwareanwendung, die Studierenden Aufgaben zuweist und die Differenzen deren UML-Klassendiagramme automatisch detektiert und kommuniziert (vgl. Soler u. a., 2010). Sowohl die Aufgaben, als auch die Musterlösungen werden von Dozierenden bereitgestellt.

- *UMLGrade* ist eine Softwareanwendung, die Studierenden Anforderungsbeschreibungen bereitstellt (vgl. Farias und Silva, 2020). Anhand dieser soll ein UML-Klassendiagramm erstellt werden. Die Softwareanwendung liefert Feedback zu detektierten Unterschieden zu einer Musterlösung und gibt Hinweise, in welchen Themenbereichen die Studierenden ihre Leistungen verbessern können.

- Ramírez-Noriega u. a. (2020) stellten fest, dass die Implementierung des Model-View-Controller Patterns Studierende regelmäßig vor Herausforderungen stellt (vgl. Ramírez-Noriega u. a., 2020). Daher entwickelten sie eine Softwareanwendung, die den Studierenden die Erstellung von Entity Relationship Modellen erlaubt. Der zugehörige Model-View-Controller Programmcode wird automatisiert generiert.

- *CAQAS* ist eine Softwareanwendung, die auf große Kurse ausgelegt ist (vgl. Knobloch u. a., 2018). Studierende können zu jeder Skript-Folie

Fragen stellen, aber auch beantworten. Der Server speichert diese Informationen gemeinsam mit einer Präsentationsfolie, zu der die Frage auftrat. Dozierende können korrekte Antworten als solche markieren.

- Jurgelaitis u. a. (2019) entwickelten unter Zuhilfenahme der Lernplattform Moodle einen gamifizierten Software Engineering-Kurs mit unterschiedlichen Leveln (vgl. Jurgelaitis u. a., 2019). Er diente vor allem dazu, UML Syntax und Semantik zu unterrichten.

- Kurkovsky (2015) präsentierte unterschiedliche Fallstudien zum Einsatz von LEGO Serious Play[13] zur Lehre von UML (vgl. Kurkovsky, 2015).

- Cai u. a. (2013) implementierten eine Anwendung, die es Studierenden ermöglicht, mithilfe von strukturierten Matrizen, ihre erstellten Softwarearchitekturen zu evaluieren (vgl. Cai u. a., 2013).

- Py u. a. (2013) präsentieren eine Softwareanwendung, die Studierenden das Markieren wichtiger Elemente in den Anforderungsbeschreibungen erlaubt. Anschließend können sie ein UML-Diagramm modellieren (vgl. Py u. a., 2013). Die Softwareanwendung hilft den Studierenden, ihre eigene Arbeit zu evaluieren.

Folgende Publikationen konnten durch den Autor in dem Zeitraum zwischen 1. Januar 2022 und 15. Mai 2023 zusätzlich identifiziert werden. Die Ein- und Ausschlusskriterien sowie die zusammengesetzten Suchbegriffe bleiben unverändert. Es erfolgte lediglich eine Anpassung des Suchzeitraums (durch den Snowballing-Prozess ist es möglich, dass Publikationen hinzukommen, die vor dem 1. Januar 2022 veröffentlicht wurden):

- Der von Fernandes und Amaral (2022) vorgestellte Ansatz nimmt studentische Aktivitäts-, Sequenz- und Zustandsdiagramme entgegen und wandelt diese in eine interne Repräsentation um. Für jeden Diagrammtyp entwickelten sie einen eigenen Workflow, der bei Ausführung durchlaufen wird. Wie bei einer dynamischen Codeprüfung erfolgt eine Simulation der internen Repräsentation und wird mit den

[13] https://www.lego.com/en-de/themes/serious-play

Vorgaben von Lehrpersonen verglichen. Detektierte Differenzen stellen das Feedback für Studierende dar (vgl. Fernandes und Amaral, 2022).

- Die von Singh u. a. (2022) veröffentlichte Softwareanwendung ermöglicht Lehrpersonen wie Studierenden die Erstellung von UML-Klassendiagrammen in einem Editor. Die resultierenden Diagramme werden in eine interne Repräsentation überführt und sind dadurch miteinander vergleichbar. Zusätzlich erfolgt eine Einordnung studentischer Fehler in vordefinierte Kategorien. Studierende erhalten eine Übersicht ihrer Fehler und der jeweiligen Kategorien als Feedback (vgl. Singh u. a., 2022).

- Bian u. a. (2021) präsentieren eine Softwareanwendung, die studentische UML-Klassendiagramme mit Musterlösungen vergleicht und anhand eines Bewertungsalgorithmus eine automatisierte Benotung ermöglicht. Dabei berücksichtigt der Algorithmus auch Synonyme und Schreibfehler in studentischen Diagrammen (vgl. Bian u. a., 2021).

- *AutoER* ist eine Softwareanwendung für die manuelle oder automatisierte Generierung und Evaluation von UML Datenbankdesignproblemen (vgl. Foss u. a., 2022). Eine Klasse repräsentiert dabei eine Datenbankentität. Die Anwendung stellt Studierenden eine textuelle Anforderungsbeschreibung bereit. Einzelne Textpassagen lassen sich markieren und deren Typ (beispielsweise Klasse oder Attribut) bestimmen. Zusätzlich erhalten sie durch Dozierende oder automatisiert generierte Fragestellungen zu den jeweiligen Aufgaben. Die Erstellung eigener UML-Klassendiagramme durch Studierende ist nicht möglich.

- Yigitbas u. a. (2022) beschreiben eine Virtual Reality Umgebung, die Studierenden zwei Bereiche zur Verfügung stellt. Bei Ersterem handelt es sich um eine gamifizierte Lernumgebung, in der Vorlesungsstoff durch Spiele (beispielsweise Galgenmännchen) wiederholt werden kann. Der zweite Bereich visualisiert ein UML-Klassendiagramm und ein zugehöriges 3D Modell der daraus resultierenden Software. Dabei handelt es sich um ein von Studierenden modifizierbares Beispiel (vgl. Yigitbas u. a., 2022).

- Feichas und Seabra (2023) präsentieren in ihrer Publikation eine interaktive, web-basierte Lernplattform für Studierende. Diese können sich dort mit vorgegebenen Lehrinhalten auseinandersetzen und anschließend Übungsaufgaben (beispielsweise Quiz-Aufgaben oder Lückentextaufgaben) lösen. Weiterhin können sie ihren Fortschritt bei unterschiedlichen Themenschwerpunkten nachvollziehen. Für Lehrpersonen gibt es ebenfalls einen Zugang, über welchen sie die Fortschritte aller ihrer Studierenden überprüfen können (vgl. Feichas und Seabra, 2023).

- Huber und Hagel (2022a) stellen in ihrer Publikation einen Ansatz vor, der eine automatisierte und durch Deep Learning gestützte Erstellung textueller Anforderungsbeschreibungen für die hochschulische Lehre von UML-Klassendiagrammen ermöglicht. Das Resultat stellt eine LaTeX-Datei und ein zugehöriges PDF dar. Diese beinhalten die automatisiert generierten Aufgabentexte und eine Aufgabenbeschreibung (vgl. Huber und Hagel, 2022a).

- Die von Huber und Hagel (2022c) präsentierte Softwareanwendung ist in der Lage, textuelle Anforderungsbeschreibungen für die hochschulische Lehre von UML-Klassendiagrammen zu importieren und anhand dieser Musterlösungen zu erzeugen. Für die Analyse der Textbausteine und die Extraktion von Diagrammkomponenten findet ein NLP-Modell für fortgeschrittene Textanalyse Anwendung. Als Endprodukt liefert die Software eine Extensible Markup Language (XML) Datei, die in dem Modellierungseditor Enterprise Architect[14] importierbar ist (vgl. Huber und Hagel, 2022c).

SLR-F2: Welche fundamentalen Differenzen bestehen zwischen diesen Softwareanwendungen?

Diese Forschungsfrage ist mithilfe der Ergebnisse von Forschungsfrage SLR-F1 beantwortbar und ist nach Huber und Hagel (2022b) in Kategorien unterteilbar.

[14]Einsehbar unter: https://sparxsystems.com/

Die beschriebenen Softwareanwendungen für eine computergestützte Hochschullehre von Software Engineering und im speziellen UML-Diagrammen unterscheiden sich in ihrer technologischen Basis und den Funktionalitäten. Mit letzteren versuchen die jeweiligen Autoren unterschiedliche, sich teils überschneidende studentische Probleme zu überwinden.

Der Fokus bei der Beantwortung von Forschungsfrage SLR-F2 liegt jedoch auf dem methodischen Ansatz, der für die Softwareanwendungen gewählt wurde. So lassen sich drei grundlegend unterschiedliche Cluster in den beschriebenen Publikationen identifizieren (die Einordnung erfolgt in Anlehnung an (Huber und Hagel, 2022b, S. 1407) und wurde um die zwischen dem 01.01.2022 und dem 15.05.2023 detektierten Publikationen erweitert). Besagte Gruppierungen sind auf Basis einer induktiven Zusammenführung vergleichbarer Vorgehensweisen bei den Softwareanwendungen entstanden und wurden anhand dessen von Huber und Hagel (2022b) gebildet. Folgende Cluster ergeben sich daraus (die prozentualen Angaben sind gerundet):

- **Visualisieren:** 19,5% der beschriebenen Publikationen fokussieren sich auf die Visualisierung von UML-Diagrammen und ggf. zugehörigem Programmcode. Die Schaffung neuer Perspektiven und Verknüpfung unterschiedlicher Zusammenhänge können Studierenden das Erlernen der UML erleichtern.

- **Vergleichen:** 58,5% der angeführten Publikationen vergleichen studentische UML-Diagramme mit Domänenwissen, vordefinierten Regelwerken oder Musterlösungen. Prozentual wurde diese Vorgehensweise erkennbar häufiger gewählt. Sie bietet Studierenden die Möglichkeit eines formativen und meist zeitnahen Feedbacks zu ihren UML-Diagrammen. Bestehenden Wissenslücken und Unklarheiten kann dadurch gezielt und individuell entgegen gewirkt werden. Zudem reduzieren diese Softwareanwendungen den Aufwand von Dozierenden.

- **Andere:** 22,0% der selektierten Publikationen lassen sich nicht in die zuvor beschriebenen Cluster sortieren. Sie unterscheiden sich in ihrem methodischen Vorgehen von den darin enthaltenen Softwareanwendungen, aber auch voneinander.

Beschreibung des Defizits	**Genannt von (in %)**
Komplex im Erlernen und in der Anwendung	63,4
Hoher Zeitaufwand für Lehrpersonen	39,0
Vorhandene Softwareanwendung reichen nicht aus	36,6
Individuelles und direktes Feedback zu geben, ist nicht möglich	31,7
Unterschiedliche Lösungen können korrekt sein	29,3

Tabelle 3.3: Auflistung von Defiziten in der Hochschullehre von UML-Diagrammen. Eigene Darstellung und Erweiterung nach (Huber und Hagel, 2022b, S. 1408).

SLR-F3: Welche Defizite sollen durch den Einsatz dieser Softwareanwendungen eliminiert werden?

Tabelle 3.3 beinhaltet eine Auflistung der von Dozierenden und Forschenden in den selektierten Publikationen beschriebenen, wichtigsten Defiziten, die in der Lehre von UML-Diagrammen bestehen. Die Tabelle wurde in Anlehnung an (Huber und Hagel, 2022b, S. 1408) durch den Autor erstellt, übersetzt und erweitert.

Das am häufigsten in allen Publikationen genannte Defizit (63,4%) von Studierenden hängt mit der Komplexität der UML zusammen. Das Erlernen der Syntax sowie die Anwendung des erworbenen Wissens auf eine textuelle Anforderungsbeschreibung stellt eine Herausforderung für Studierende dar (vgl. Huber und Hagel, 2022b). Hinzu kommt die Komplexität, die mit objekt-orientierten Konzepten einhergeht.

39,0% der identifizierten Veröffentlichungen beschreiben den hohen Zeitaufwand, der für Lehrpersonen bei der Erstellung von Übungsaufgaben und zugehörigen Musterlösungen einhergeht. Besonders aufwendig ist jedoch die Bereitstellung von individuellem und direktem Feedback für Studierende (vgl. Huber und Hagel, 2022b).

Weiterhin nennen 36,6% vorhandene Softwarelösungen als Defizit. Diese seien laut der Autoren (nachzuschlagen bei Huber und Hagel (2022b) oder in den angeführten Publikationen) für die hochschulische Lehre von UML-Dia-

grammen nicht ausreichend geeignet. So bieten solche Anwendungen häufig entweder einen zu großen Funktionsumfang oder benötigte Funktionalitäten sind nicht vorhanden.

Dass eine Verfügbarkeit von individuellem und direktem Feedback aus zeitlichen Gründen nicht möglich ist, wurde von 31,7% der Publikationen angeführt. Aufgrund der Vielfalt an Lösungsmöglichkeiten für eine gegebene Modellierungsaufgabe (dieses Defizit findet sich in 29,3% der identifizierten Veröffentlichungen) müssen Lehrpersonen jedes studentische Diagramm individuell bewerten.

3.2 Anforderungserhebung anhand der Literaturübersicht

Die durch den vorherigen Abschnitt beschriebenen Publikationen implizieren Lehrsituationen, in denen Studierende Hilfestellungen respektive Feedback durch Dozierende oder ein computergestütztes System erhalten (siehe Feedback Pfade in Tabelle 2.1). Diese Ergebnisse verdeutlichen, dass es zwei Gruppen von Akteuren gibt, die mit den computergestützten Softwareanwendungen interagieren: Dozierende und Studierende. Erstere nutzen sie, um Letzteren Aufgabenstellungen, Visualisierungen oder automatisiertes Feedback zur Verbesserung ihres Lernerfolgs bereitzustellen. Der dabei gewählte Feedback Typ (siehe Tabelle 2.2) variiert.

Da in dieser Arbeit eine computergestützte Softwareanwendung zur Lehre von UML-Klassendiagrammen in der hochschulischen Lehre von Software Engineering konzipiert und prototypisch implementiert werden soll, sind Anforderungen mithilfe der Akteure zu erheben. Die daraus resultierenden Funktionalitäten wirken neben weiteren Aspekten den in dem vorherigen Abschnitt beschriebenen Defiziten gezielt und individualisiert entgegen. Die Identifikation funktionaler Systemanforderungen ist unabdingbar, da sie wichtige Aktionen beschreiben, die das zu konzipierende System den Akteuren bereitstellt. Als Grundlage dienen die in der SLR angeführten Publikationen. Sie werden dem DSR-Forschungsdesign nach Hevner u. a. (2004) zufolge als Wissensbasis herangezogen und sind eine Grundlage für das entstehende Artefakt.

In Anlehnung an Huber u. a. (2023a) erfolgt eine Untergliederung der funktionalen Systemanforderungen im Folgenden nach den jeweiligen Akteuren sowie Funktionalitäten, die anwenderunabhängig bereitgestellt werden sollen sowie nach methodischem Ansatz (Visualisieren oder Vergleichen; siehe Abschnitt 3.1.3). Sofern eine Funktionalität in drei oder mehr unterschiedlichen Publikationen beschrieben ist, fand sie Einzug in die Auflistung. Systemanforderungen für Visualisierungen finden sich in separaten Tabellen (vgl. Huber u. a., 2023a), wohingegen Systemanforderungen für den Vergleich studentischer Diagramme mit Domänenwissen, Regeln oder Musterlösungen als Basisanforderungen angesehen werden, da dieses Vorgehen in der Literatur deutlich häufiger gewählt wurde.

Als Entwicklungsgrundlage für das zu konzipierende Artefakt wird sich im Sinne des DSR-Forschungsdesigns nach Hevner u. a. (2004) erneut der Wissensbasis durch Auswahl eines geeigneten Dokumentationsverfahrens für Anforderungen bedient. In Anlehnung an Huber u. a. (2023a) finden die Vorgaben von Rupp u. a. (2004) Anwendung. Mithilfe unterschiedlicher Spezifikationslevel sind Anforderungen mit unterschiedlichen Granularitäten definierbar. Laut der Autoren stehe bei einer Systemanalyse jedoch das Spezifikationslevel 2 im Fokus (vgl. Rupp u. a., 2014). Alle innerhalb der Arbeit erhobenen Anforderungen orientieren sich an dieser Granularität.

Tabelle 3.4, 3.5 und 3.6 beinhaltet Basisanforderungen für Dozierende, Studierende und Systemanforderungen. Tabelle 3.7 und 3.8 fassen Anforderungen zur Visualisierung für Studierende und Systemanforderungen zusammen. Dozierende erhalten keine separate Tabelle für Visualisierungen, da der Wissensbasis diesbezüglich keine Punkte entnommen werden konnten. Eindeutig identifizierbar sind diese anhand der frei gewählten Kürzel „BDA“, „BSA“, „BSYA“, „VSA“ und „VSYA“ sowie einer eindeutigen Zahl (gegebenenfalls in Kombination mit einem Buchstaben, der eine alphabetische Sortierung erlaubt).

ID	**Anforderung**
BDA1a	Das System muss Dozierenden die Möglichkeit bieten, eine textuelle Anforderungsbeschreibung für die Lehre von UML-Klassendiagrammen einzugeben.

BDA1b	Das System muss Dozierenden die Möglichkeit bieten, eine textuelle Anforderungsbeschreibung für die Lehre von UML-Klassendiagrammen zu speichern.
BDA2a	Das System muss Dozierenden die Möglichkeit bieten, textuelle Informationen über das Lernziel einer zugehörigen, textuellen Anforderungsbeschreibung einzugeben.
BDA2b	Das System muss Dozierenden die Möglichkeit bieten, textuelle Informationen über das Lernziel einer zugehörigen, textuellen Anforderungsbeschreibung zu speichern.
BDA3	Das System muss Dozierenden die Möglichkeit bieten, mehreren Studierenden zur gleichen Zeit eine textuelle Anforderungsbeschreibung bereitzustellen.
BDA4a	Das System muss Dozierenden die Möglichkeit bieten, eine Musterlösung in Form einer Bilddatei (im JPG- oder PNG-Format) oder einer XML-Datei für eine textuelle Anforderungsbeschreibung hochzuladen.
BDA4b	Das System muss Dozierenden die Möglichkeit bieten, eine Musterlösung in Form einer Bilddatei (im JPG- oder PNG-Format) oder einer XML-Datei für eine textuelle Anforderungsbeschreibung zu speichern.
BDA5a	Das System sollte Dozierenden die Möglichkeit bieten, eine Musterlösung durch einen UML-Editor für eine textuelle Anforderungsbeschreibung zu erstellen.
BDA5b	Das System sollte Dozierenden die Möglichkeit bieten, eine Musterlösung durch einen UML-Editor für eine textuelle Anforderungsbeschreibung zu speichern.

BDA6	Das System sollte Dozierenden die Möglichkeit bieten zu spezifizieren, welche Komponenten eines UML-Klassendiagramms das System bei einem Vergleich berücksichtigen soll.
BDA7	Das System sollte Dozierenden die Möglichkeit bieten, Fragen von Studierenden zu einer textuellen Anforderungsbeschreibung in einem Chat-Fenster einzusehen.
BDA8	Das System sollte Dozierenden die Möglichkeit bieten, Fragen von Studierenden zu einer textuellen Anforderungsbeschreibung in einem Chat-Fenster zu beantworten.
BDA9	Das System sollte Dozierenden die Möglichkeit bieten, eine statistische Übersicht über die Anzahl an Fehlern in studentischen UML-Klassendiagrammen einzusehen.
BDA10	Das System sollte Dozierenden die Möglichkeit bieten, die von Studierenden hochgeladenen UML-Klassendiagramme zu einer textuellen Anforderungsbeschreibung und deren Differenzen zu einer Musterlösung einzusehen.

Tabelle 3.4: Basisanforderungen für Dozierende. Eigene Darstellung und Erweiterung nach (Huber u. a., 2023a, S. 185).

ID	**Anforderung**
BSA1	Das System muss Studierenden die Möglichkeit bieten, eine textuelle Anforderungsbeschreibung einzusehen.

BSA2	Das System muss Studierenden die Möglichkeit bieten, die von Dozierenden bereitgestellte Information über das Lernziel einer textuellen Anforderungsbeschreibung einzusehen.
BSA3	Das System muss Studierenden die Möglichkeit bieten, UML-Klassendiagramme mithilfe eines UML-Editors zu modellieren.
BSA4	Das System muss Studierenden die Möglichkeit bieten, Fragen zu einer textuellen Anforderungsbeschreibung in einem Chat-Fenster zu stellen.
BSA5	Das System sollte Studierenden die Möglichkeit bieten, Feedback für ein UML-Klassendiagramm während des Modellierungsvorgangs zu erhalten.
BSA6	Das System sollte Studierenden die Möglichkeit bieten, die in textuellen Anforderungsbeschreibungen enthaltenen Diagrammbestandteile anzuzeigen.
BSA7	Das System sollte Studierenden die Möglichkeit bieten, UML-Klassendiagramme in XML-, JPG- oder PNG-Format hochzuladen.
BSA8	Das System sollte Studierenden die Möglichkeit bieten, ein UML-Klassendiagramm an einen Server zu schicken.
BSA9	Das System sollte Studierenden die Möglichkeit bieten, die während eines Vergleichs zweier UML-Klassendiagramme detektierten Differenzen einzusehen.

BSA10	Nachdem zwei UML-Klassendiagramme miteinander verglichen wurden, sollte das System eine Liste mit weiterführender Literatur bereitstellen, die Studierende für die Verbesserung ihrer Leistung einsehen können.
BSA11	Nachdem die Studierenden Feedback durch das System erhalten haben, sollte das System Studierenden die Möglichkeit bieten, eine verbesserte Version ihres UML-Klassendiagramms hochzuladen.

Tabelle 3.5: Basisanforderungen für Studierende. Eigene Darstellung und Erweiterung nach (Huber u. a., 2023a, S. 186).

ID	**Anforderung**
BSYA1	Das System sollte studentische Aktivitäten während der Modellierung eines UML-Klassendiagramms loggen.
BSYA2	Das System sollte modellierte UML-Klassendiagramme in eine XML-Datei exportieren.
BSYA3	Das System muss hochgeladene UML-Klassendiagramme in eine interne Struktur überführen, die Vergleiche zwischen mehreren UML-Klassendiagrammen erlaubt.
BSYA4	Das System muss studentische UML-Klassendiagramme anhand eines Bewertungsalgorithmus benoten.
BSYA5	Das System sollte Synonyme für Klassen-, Attribut- und Methodennamen in der Bewertung berücksichtigen.
BSYA6	Das System sollte Statistiken über die Anzahl detektierter Fehler in studentischen UML-Klassendiagrammen speichern.

BSYA7	Nach einem Vergleich zweier UML-Klassendiagramme muss das System Feedback bereitstellen.
BSYA8	Das System muss zwei UML-Klassendiagramme miteinander vergleichen und deren Differenzen detektieren.
BSYA9	Das System sollte studentische UML-Klassendiagramme mit einem vordefinierten Regelwerk vergleichen und Abweichungen detektieren.
BSYA10	Das System muss automatisiert textuelle Anforderungsbeschreibungen generieren.
BSYA11	Das System muss Musterlösungen in Form einer XML-Datei für eine vorgegebene textuelle Anforderungsbeschreibung erzeugen.

Tabelle 3.6: Basis-Systemanforderungen. Eigene Darstellung und Erweiterung nach (Huber u. a., 2023a, S. 186).

ID	**Anforderung**
VSA1	Das System sollte Studierenden die Möglichkeit bieten, Programmcode für eine textuelle Anforderungsbeschreibung zu schreiben.
VSA2	Das System sollte Studierenden die Möglichkeit bieten, Programmcode in transformierter Form als UML-Klassendiagramm einzusehen.
VSA3	Das System sollte Studierenden die Möglichkeit bieten, automatisiert erzeugten Programmcode zu einem modellierten UML-Klassendiagramm einzusehen.

VSA4	Während der Modellierung mit einem bereitgestellten UML-Editor sollte das System Studierenden die Möglichkeit bieten, nach Schlüsselworten innerhalb des UML-Klassendiagramms zu suchen.
VSA5	Das System sollte Studierenden die Möglichkeit bieten, ein 2D UML-Klassendiagramm in Augmented Reality Ansicht, 3D oder Virtual Reality Ansicht einzusehen.
VSA6	Das System sollte Studierenden die Möglichkeit bieten, erstellten Programmcode auszuführen und die Veränderungen der Objektparameter zur Laufzeit anzuzeigen.
VSA7	Das System sollte Studierenden die Möglichkeit bieten, mit erstellten UML-Klassendiagrammen in 2D, Augmented Reality, 3D oder Virtual Reality zu interagieren.

Tabelle 3.7: Visualisierungsanforderungen für Studierende. Eigene Darstellung und Erweiterung nach (Huber u. a., 2023a, S. 186).

ID	**Anforderung**
VSYA1	Falls Studierende ein 3D UML-Klassendiagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein 3D UML-Klassendiagramm zu konvertieren.
VSYA2	Das System sollte verschiedene Informationsebenen (Klassen, Attribute, Methoden) einblenden.
VSYA3	Falls Studierende ein Augmented Reality Diagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein Augmented Reality Diagramm zu konvertieren.

VSYA4	Das System sollte Programmcode neben einem zugehörigen UML-Klassendiagramm anzeigen.
VSYA5	Falls Studierende ein Virtual Reality UML-Klassendiagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein Virtual Reality UML-Klassendiagramm konvertieren.
VSYA6	Falls Studierende im Programmcode eine Zeile markieren, sollte das System zugehörige Diagrammbestandteile in dem zugehörigen UML-Klassendiagramm hervorheben.
VSYA7	Das System sollte Lehrinhalte in Form von PDF-Dateien darstellen.

Tabelle 3.8: Visualisierungsanforderungen für das System. Eigene Darstellung und Erweiterung nach (Huber u. a., 2023a, S. 187).

Anhand der beschriebenen Anforderungen lassen sich Aktivitäten ableiten, die von den Akteuren innerhalb des Systems durchführbar und im Folgenden beschrieben sind. Die in Anhang A.1 enthaltenen UML-Aktivitätsdiagramme fassen diese aus der Sicht von Dozierenden (siehe Abbildung A.1) und Studierenden (siehe Abbildung A.2) zusammen. Bei beiden Diagrammen und den zugehörigen Beschreibungen im Folgenden handelt es sich um eine eigene Darstellung in Anlehnung an Huber u. a. (2023a). Es erfolgte eine Übersetzung der Inhalte durch den Autor in die deutsche Sprache.

Nach Start der Softwareanwendung können Dozierende eine für sie vorgesehene Oberfläche aufrufen. Darin stehen ihnen Eingabefelder für das zu verfolgende Lernziel und die textuellen Anforderungsbeschreibung bereit. Musterlösungen können hochgeladen oder in einem bereitstehenden Editor modelliert werden. Nach Wahl der in einem Diagrammvergleich zu berücksichtigenden Komponenten, ist die Übung startbereit. Während des Modellierungsvorgangs durch Studierende können Dozierende aufkommende Fragen über eine Chat-Funktion beantworten. Sind alle Modellierungen abgeschlossen, sehen die Dozierenden Statistiken über die Aktivitäten der Studierenden sowie deren abgegebenen Diagramme und Bewertungen durch

die Softwareanwendung. Letztlich wird die Übung beendet.

Studierende hingegen sehen sowohl die von Dozierenden bereitgestellte Übungsaufgabe als auch das angestrebte Lernziel. Ein UML-Klassendiagramm ist mithilfe einer externen Modellierungsanwendung oder einem internen Editor generierbar. Während des Modellierungsvorgangs können Studierende diverse Hilfsfunktionen (beispielsweise einen Chat für Fragen) oder Visualisierungen in Anspruch nehmen. Nach Fertigstellung und Prüfung des UML-Klassendiagramms erfolgt die Abgabe mit anschließender Bereitstellung von Feedback. Nach Wahl können die Studierenden eine verbesserte Version ihres Diagramms erstellen oder die Übung beenden.

3.3 Qualitative Erhebung von Anforderungen der Studierenden

Eine Untersuchung der in der Literatur beschriebenen Funktionalitäten von Softwareanwendungen zur Unterstützung der hochschulischen Lehre von UML-Diagrammen ist eine wichtige Grundvoraussetzung für die Entwicklung eines neuen Artefakts. Dozierende und Forschende beschreiben in ihren Publikationen meist Defizite, anhand derer Funktionalitäten abgeleitet wurden (vgl. Huber und Hagel, 2022b). Weiterhin ist jedoch eine Untersuchung aller studentischen Anforderungen notwendig, um ihren Lernprozess optimal unterstützen zu können. Da die bestehende Wissensbasis nicht hinreichend Aufschluss über diese geben konnte, führten Huber u. a. (2023b) qualitative, leitfadengestützte Experteninterviews mit Studierenden durch. Auch die daraus erhobenen Anforderungen dienen als essenzielle Grundlage für die Konzipierung und prototypische Implementierung des Artefakts.

Das Forschungsdesign der Experteninterviews von Huber u. a. (2023b) orientiert sich an den Spezifikationen von Baur und Blasius (2019) sowie Bogner u. a. (2014). Die Struktur der folgenden Unterabschnitte ist daran angelehnt.

3.3.1 Forschungsdesiderat und -fragen

Als wichtigstes Forschungsdesiderat führen Huber u. a. (2023b) die Erhebung studentischer Anforderungen an eine Software für die Unterstützung

der hochschulischen Lehre von UML-Diagrammen an. Aus diesem Ziel leiten sie drei Themenschwerpunkte ab, welche für die Struktur der Experteninterviews maßgebend sind. In einem ersten Schritt wollten die Autoren in Erfahrung bringen, mit welchen Herausforderungen sich Studierende beim Erlernen von UML-Klassendiagrammen konfrontiert sehen (vgl. Huber u. a., 2023b). Über diese gibt auch die Literatur umfassend Aufschluss (vgl. Huber u. a., 2023b; vgl. Huber und Hagel, 2022b). Dieser Teil des Experteninterviews soll Studierenden als Möglichkeit zur Reflexion dienen und erste Denkanstöße für potenzielle computergestützte Hilfestellungen geben.

Aus dem Forschungsdesiderat leiten Huber u. a. (2023b) die zwei Folgenden, durch den Autor übersetzten, Forschungsfragen ab (Huber u. a., 2023b, S. 190):

- **SIF1:** Mit welchen Herausforderungen sehen sich Studierende beim Erlernen von UML-Modellen im Kontext der hochschulischen Lehre von Software Engineering konfrontiert?
- **SIF2:** Welche softwaregestützten Hilfestellungen können Studierende dabei unterstützen, die von ihnen genannten Herausforderungen bei der Erstellung von UML-Diagrammen anhand textueller Aufgaben zu überwinden?

Das Kürzel „SIF“ ist dabei frei gewählt und dient lediglich der eindeutigen Identifikation der Forschungsfragen.

3.3.2 Struktur des qualitativen, leitfadengestützten Experteninterviews

Da eine Extraktion der gewünschten Informationen anhand der bestehenden Literatur nicht möglich war, wählten Huber u. a. (2023b) ein qualitatives, leitfadengestütztes Experteninterview für die Erhebung von Anforderungen. Nach den Spezifikationen von Bogner u. a. (2014) gelten Studierende in diesem Rahmen als Experten. Sie sehen sich in ihrem Studienalltag wiederholt mit dem beschriebenen Problemkontext konfrontiert und haben daher Einsichten, die nur von dieser Personengruppe offengelegt werden können. Huber u. a. (2023b) entwickelten einen Leitfaden für das Experteninterview nach den Angaben von Baur und Blasius (2019) sowie Bogner u. a. (2014).

Dieser diente als Gesprächsgrundlage. Um den Einstieg in das Gespräch zu erleichtern (vgl. Baur und Blasius, 2019), planten die Autoren folgende drei Fragen (eigene Übersetzung nach (Huber u. a., 2023b, S. 190)):

- Wie alt sind Sie?
- In welchem Fachsemester sind Sie?
- Welchen Studiengang belegen Sie?

Anschließend wurden die in dem vorherigen Unterabschnitt beschriebenen Forschungsfragen gestellt (vgl. Huber u. a., 2023b). Zwischenfragen stellten die Autoren, um den Gesprächsfluss aktiv zu gestalten.

3.3.3 Teilnehmende

Huber u. a. (2023b) führten das qualitative, leitfadengestützte Experteninterview an der Hochschule für angewandte Wissenschaften in Kempten (Deutschland) durch. Als mögliche Teilnehmende kamen Studierende aus informatiknahen Bachelorstudiengängen in Frage. Als Grundvoraussetzung galt, dass alle zu befragenden Personen den Software Engineering-Kurs zum Zeitpunkt des Interviews gehört sowie dessen Übungen besucht und durchgeführt hatten. Die Befragung erfolgte mit fünf Interviewpartnerinnen und -partnern im Wintersemester 2022 / 2023. Bei einer qualitativen Erhebung ist die Zahl der Teilnehmenden üblicherweise geringer als bei quantitativen Studien (vgl. Bogner u. a., 2014; vgl. Baur und Blasius, 2019). Da durch das fünfte geführte Interview keine Signifikanten neuen Erkenntnisse mehr feststellbar waren, ist die Anzahl der befragten Personen als ausreichend einzustufen.

3.3.4 Extraktion von Anforderungen

Wie von Bogner u. a. (2014) beschrieben, ist die qualitative Inhaltsanalyse ein geeignetes Instrumentarium für die Auswertung von Experteninterviews (vgl. Bogner u. a., 2014). Dabei handelt es sich um eine streng regelbasierte, intersubjektiv überprüfbare Auswertungsmethode für Texte (vgl. Mayring und Fenzl, 2022). Ihr Ziel ist es, die Interpretation von qualitativ erhobenem

Textmaterial mithilfe reglementierter Verfahrensschritte zu ermöglichen und überprüfbar zu machen (vgl. Mayring, 2010). Sie ist somit von freien und subjektiven Interpretationstechniken abzugrenzen.

Die qualitative Inhaltsanalyse bietet drei unterschiedliche Grundtechniken (in Anlehnung an (vgl. Mayring, 2008; vgl. Mayring, 2010; vgl. Mayring und Fenzl, 2022)):

- **Zusammenfassung:** Dient der Reduktion von Inhalten auf deren Kernaussagen. Durch Definition eines Selektionskriteriums können Textbausteine ausgewählt und durch ein regelbasiertes Vorgehensmodell zu induktiv gebildeten oder deduktiv beschriebenen Kategorien zugeordnet werden.

- **Explikation**: Zieht einzelne Textstellen heran, die unklar oder nicht verständlich sind und versucht diese mithilfe von kontextuellen Rückschlüssen verständlich zu machen.

- **Strukturierung:** Dient der Extraktion wichtiger Aspekte aus Textbausteinen. Dazu werden vorab deduktive Kategorien definiert, denen die jeweiligen Textstellen zuzuordnen sind.

Für die Beantwortung der Forschungsfragen haben Huber u. a. (2023b) die zusammenfassende Inhaltsanalyse als Interpretationstechnik zugrunde gelegt. Die in Abbildung 3.2 ersichtlichen Schritte visualisieren ein dafür von Mayring und Brunner (2006) vorgeschlagenes Ablaufmodell. Diese Schritte wurden auch von Huber u. a. (2023b) durchgeführt. Die folgende Auflistung beschreibt die Inhalte der jeweiligen Schritte und spezifiziert, wie diese im Beispiel von Huber u. a. (2023b) angewendet wurden (in Anlehnung an: (vgl. Mayring, 2010; vgl. Mayring und Fenzl, 2022)):

- **Schritt 1:** Dient der Beschreibung des zugrundeliegenden Forschungsdesiderats und präzisiert die Forschungsfragen. Diese sind in Unterabschnitt 3.3.1 der Arbeit erläutert und werden daher an dieser Stelle nicht erneut aufgegriffen.

- **Schritt 2:** Beinhaltet die Auswahl und Charakterisierung der zu untersuchenden Texte. Bei Huber u. a. (2023b) handelt es sich um die Transkripte der einzelnen Experteninterviews.

- **Schritt 3:** Ordnet die Texte in ein Kommunikationsmodell ein. Bei diesem werden „[...] Textproduzenten, sozio-kultureller Hintergrund, Textproduktionssituation, Textwirkungen und Zielgruppen des Textes eruiert [...]“ (Mayring und Fenzl, 2022, S. 636). Die genannten Punkte wurden von Huber u. a. (2023b) beschrieben und sind in den vorangegangenen Unterabschnitten der Arbeit näher beleuchtet.

- **Schritt 4:** Bestimmt die Analyseeinheiten (auch Kodiereinheiten genannt). Sie sind der kleinstmögliche Textbestandteil, der zur Auswertung bereitsteht. Bei Huber u. a. (2023b) stellt ein einzelner Satz die kleinste Analyseeinheit dar.

- **Schritt 5 (Induktiv):** Bestimmt, wie Kategorien mithilfe eines induktiven Verfahrens gebildet werden können beziehungsweise was diese ausmacht. Eine Kategorie ist bei Huber u. a. (2023b) eine Gruppierung von Herausforderungen oder Anforderungen, die sich zwar individuell voneinander unterscheiden, jedoch auf eine gemeinsame Herausforderung oder einen gemeinsamen Lösungsvorschlag hinweisen. Die Abstraktionsebene liegt dabei eine Ebene über den konkreten Antworten der Interviews.

- **Schritt 5 (Deduktiv):** Legt theoriegeleitet fest, was eine Kategorie ausmacht, der die einzelnen Analyseeinheiten zugeordnet werden können.

- **Schritt 6 (Induktiv):** Definiert, wie allgemein bzw. abstrakt eine Kategorie formuliert werden darf. Das von Huber u. a. (2023b) präsentierte Abstraktionsniveau bestimmt, dass konkrete Aussagen in den einzelnen Analyseeinheiten zu einer gemeinsamen, übergeordneten Gruppierung zuordenbar sind. Die Kategorien weisen auf übergeordnete Herausforderungen oder Anforderungswünsche hin. Eine weitere Gruppierung der Kategorien ist nicht erfolgt, da das Abstraktionsniveau somit zu hoch und die daraus entstehenden Oberkategorien zu allgemein gehalten wären.

- **Schritt 6 (Deduktiv):** Erstellt einen Leitfaden der festlegt, wie einzelne Kategorien definiert sind, wie Textbausteine dieser Kategorien

aussehen können und welche Kodierregeln zur Differenzierung anderer Kategorien es gibt.

- **Schritt 7 (Induktiv):** Führt die induktive Kategoriebildung anhand der zuvor festgelegten Parameter durch.
- **Schritt 7 (Deduktiv):** Wendet die Kategoriedefinitionen an und ordnet die Analyseeinheiten in die deduktiven Kategorien ein.
- **Schritt 8:** Dient der Überprüfung des Kategoriesystems durch eine erneute Überarbeitung des Textes (ggf. auch nur zu Teilen) ohne Berücksichtigung der zuvor erfolgten Kodierung. Der Grad der Übereinstimmung beider Ergebnisse bestimmen Güte und Belastbarkeit des Verfahrens. Auch bei Huber u. a. (2023b) fand diese Überprüfung Anwendung.
- **Schritt 9:** Beinhaltet den finalen Durchgang der Texte und Anwendung des überarbeiteten Kategoriesystems.
- **Schritt 10:** Erhöht die Stabilität des Verfahrens durch Verifikation der Ergebnisse mithilfe weiterer Personen. Diese führen den beschriebenen Prozess erneut aus und prüfen, ob dasselbe Ergebnis erzielt werden kann. Die qualitative Inhaltsanalyse erhält somit die Eigenschaft der Intersubjektivität.
- **Schritt 11:** Dient der qualitativen und quantitativen Auswertung der Kategorien.

Die Mehrstufigkeit und Reglementierungen tragen zur Komplexität des Verfahrens bei. Sie kann durch den Einsatz der von Mayring und Fenzl entwickelten, open-source Webanwendung QCAmap[15] jedoch deutlich reduziert werden. Die Anwendung leitet seine Benutzerinnen und Benutzer durch die in diesem Abschnitt beschriebenen Schritte. Auch Huber u. a. (2023b) machten von dieser Hilfestellung Gebrauch, um sicherzustellen, dass die Auswertung der leitfadengestützten Experteninterviews belastbar und nicht subjektiv ist.

Zwar sind die Herausforderungen von Studierenden bei der Modellierung von UML-Klassendiagrammen in der Literatur bereits beschrieben (siehe

[15]Einsehbar unter: https://www.qcamap.org/ui/en/home

Abschnitt 2.1.3). Dennoch ist es für diese Arbeit von großer Relevanz, diese Thematik vollumfänglich zu beleuchten und zu kategorisieren. In einem Folgeschritt dienen diese als Basis für Funktionalitäten der zu entwickelnden Software zur Unterstützung der Lehre von UML-Klassendiagrammen in der hochschulischen Lehre. Im Rahmen eines zu entwickelnden studierendenzentrierten und softwaregestützten Ansatzes sind die Anforderungen sowie Wünsche von Studierenden von zentraler Bedeutung.

In den von Huber u. a. (2023b) durchgeführten Interviews dienen die Fragen nach den Herausforderungen als Gedankenstütze, die Studierende an ihre Schwierigkeiten erinnern und sie anschließend über digitale Lösungsmöglichkeiten nachdenken lässt. Die Interviewfragen nehmen allgemeinen Bezug auf UML-Diagramme. Die in den folgenden Unterabschnitten beschriebenen Ergebnisse sind jedoch auf Klassendiagramme übertragbar und bieten zusätzliche Erkenntnisse. Alle Ausführungen und erläuterten Ergebnisse in den Unterabschnitten 3.3.5, 3.3.6 und 3.3.7 sind an Huber u. a. (2023b) angelehnt (übersetzt und sinngemäß zusammengefasst). Um den Lesefluss aufrecht zu erhalten, wird die besagte Publikation in diesen spezifischen Absätzen nicht nach jeder Aussage zitiert. In den Tabellen angeführte Identifikatoren (IDs) sind beliebig gewählt und dienen lediglich der eindeutigen Identifizierung der beschriebenen Komponenten.

3.3.5 Übersicht über die Teilnehmenden

Fünf Studierende aus den Bachelorstudiengängen Informatik und Wirtschaftsinformatik der Hochschule für angewandte Wissenschaften Kempten (Deutschland) hatten sich für ein Interview bereiterklärt. Die Altersverteilung lag zwischen 25 und 31 Jahren. Die Teilnehmenden befinden sich zwischen dem fünften und zehnten Studiensemester eines informatiknahen Bachelorstudiengangs.

Aufgrund der Verteilung im Alter, der Studiengänge und der Fachsemester gehen Huber u. a. (2023b) davon aus, dass die Studierenden unterschiedliche Erfahrungen und Wissensstände aufweisen und daher diverse Antworten respektive Einblicke zu den Interviewfragen liefern können.

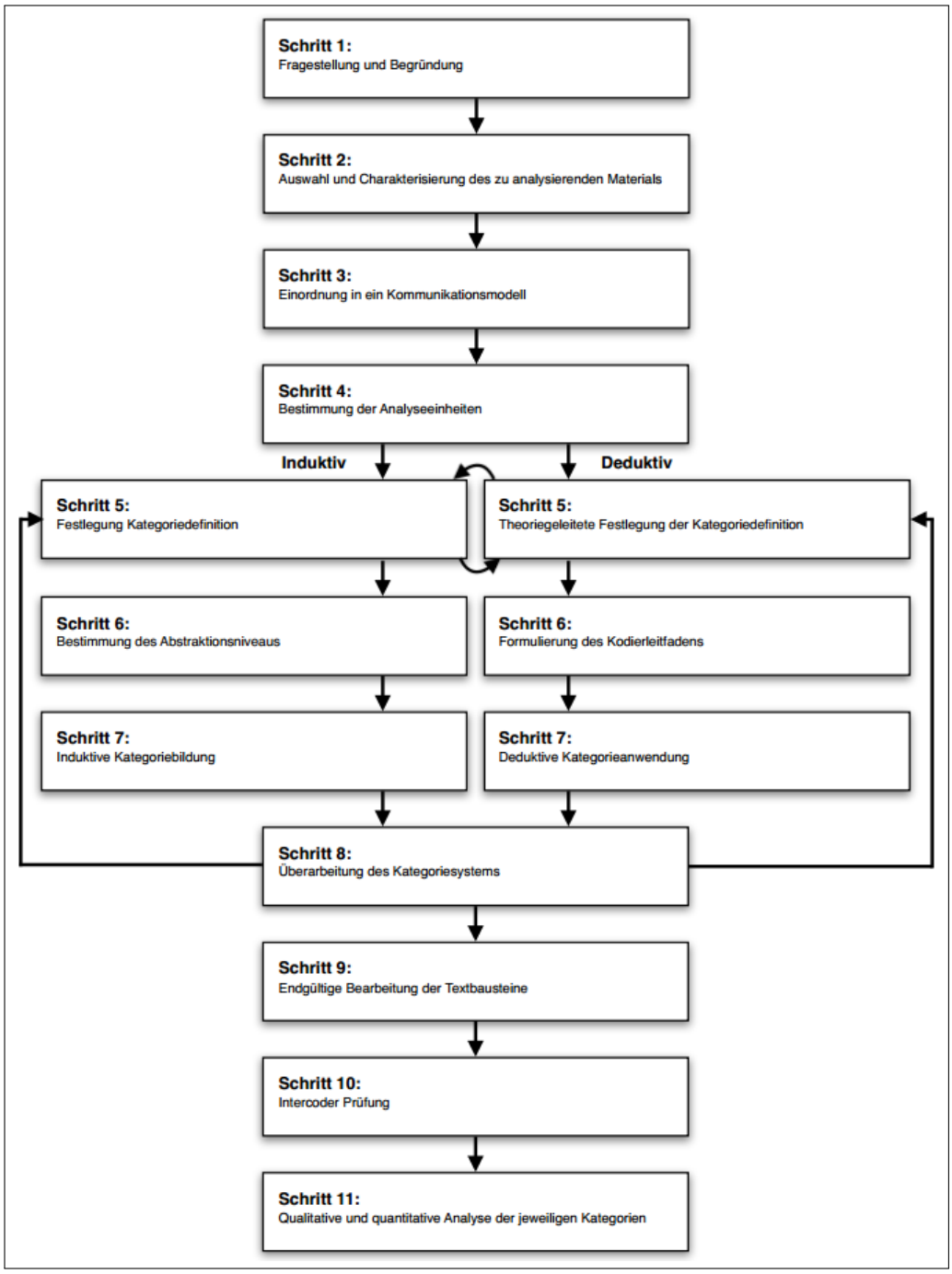


Abbildung 3.2: Ablaufmodell der qualitativen Inhaltsanalyse. Eigene Darstellung in Anlehnung an (Mayring und Brunner, 2006, S. 453-462; Mayring und Fenzl, 2022, S. 640).

ID	Kategorie	Prozent
SIF1-K1	Objektorientierung	100
SIF1-K2	Anwendung der UML	100
SIF1-K3	Komplexität der Syntax	80
SIF1-K4	Bearbeitung textueller Aufgaben	80

Tabelle 3.9: Kategorien der qualitativ erhobenen Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 191).

3.3.6 Herausforderungen für Studierende bei der Modellierung mit UML-Diagrammen

Tabelle 3.9 beinhaltet eine Auflistung der mithilfe des induktiven Verfahrens entwickelten Kategorien. Die Tabellen 3.10, 3.11, 3.12 und 3.13 listen die in Interviews erhobenen Herausforderungen der jeweiligen Kategorien auf. Die prozentualen Angaben beziehen sich (auch im folgenden Unterabschnitt) auf die genannte Häufigkeit der Kategorie oder spezifischer Herausforderungen über alle Interviews hinweg. Eine Sortierung der jeweiligen Punkte ist nach prozentualer Häufigkeit erfolgt.

Das Erlernen und Anwenden objektorientierter Konzepte stellt Studierende vor Herausforderungen. Ersichtlich ist das sowohl an den Ergebnissen des Experteninterviews von Huber u. a. (2023b), als auch anhand der Literatur (vgl. Huber und Hagel, 2022b). Dass diese Kategorie durch die induktive Vorgehensweise gebildet wurde ist daher nicht überraschend, zeigt aber eine Korrelation zu bestehender Literatur und somit eine Validität der Ergebnisse. Die am häufigsten genannte Herausforderung in dieser Kategorie ist die Abstraktheit der UML und ihren Diagrammkomponenten. Besonders zu Beginn ist es für Studierende schwierig, diese mit bestehendem Wissen, beispielsweise mit dem Klassenkonzept in einem Programmcode, zu verknüpfen. Sofern die Prinzipien der Objektorientierung noch nicht vollständig durchdrungen wurden, treten weitere Herausforderungen auf. So kommt es beispielsweise vor, dass Studierende textuell beschriebene Attribute als Klasse interpretieren. Die Überführung oder auch Abstraktion der Beschreibungen in ein UML-Diagramm zählt ebenfalls zu den Herausforderungen. Die fehlende Verknüpfung des bestehenden Wissens kann dazu führen, dass

ID	Herausforderung	Prozent
SIF1-K1-1	Abstraktheit der UML-Komponenten	60
SIF1-K1-2	Lernen und Anwenden objektorientierter Konzepte	40
SIF1-K1-3	Abstraktion von textuellen Beschreibungen	20
SIF1-K1-4	Verknüpfung mit Programmcode	20

Tabelle 3.10: Qualitativ erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Objektorientierung*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 191).

ID	Herausforderung	Prozent
SIF1-K2-1	Kein individuelles und direktes Feedback	80
SIF1-K2-2	Vielzahl der Modellierungsmöglichkeiten	60
SIF1-K2-3	Finden und Anwenden einer Strategie	60
SIF1-K2-4	Transfer des mentalen Modells in ein UML-Diagramm	40
SIF1-K2-5	Komplexität großer UML-Diagramme	20
SIF1-K2-6	Komplexität der Modellierungssoftware	20
SIF1-K2-7	Identifikation des korrekten Entwurfsmusters	20
SIF1-K2-8	Korrekte Anwendung von Entwurfsmustern	20

Tabelle 3.11: Qualitativ erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Anwendung der UML*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 191).

Studierende ihre erstellen Modellierungen nur schwer in Programmcode umwandeln können.

Auch das praktische Anwenden des theoretisch erlernten Wissens konfrontiert Studierende mit unterschiedlichen Schwierigkeiten. Allen voran die Abwesenheit von individuellem und direktem Feedback. Gerade durch die Vielzahl der Modellierungsmöglichkeiten entsteht häufig Unsicherheit, ob es sich bei dem erstellen Diagramm um eine korrekte Lösung handelt. Die Identifikation und der Einsatz einer Strategie kann bei einer fehlerfreien Erstellung von UML-Diagrammen anhand textueller Anforderungsbeschreibungen merklich unterstützen. Dies gilt sowohl für die Extraktion beschriebener Diagrammkomponenten als auch beim Transfer des mental erstellen Modells in ein UML-Diagramm. Neben der Komplexität großer Diagramme

ID	Herausforderung	Prozent
SIF1-K3-1	Vielzahl der zu erlernenden Komponenten	60
SIF1-K3-2	Differenzierung der Komponenten	60

Tabelle 3.12: Qualitativ erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Komplexität der Syntax*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 191).

wurde auch die Komplexität von Modellierungssoftware als herausfordernd beschrieben. Besagte Softwareanwendungen sind in vielen Fällen für die berufliche Praxis ausgelegt und mit einer großen Anzahl an Funktionalitäten ausgestattet. Nach eigenen Angaben waren einige der Teilnehmenden damit überfordert. Auch die Identifikation und die korrekte Anwendung von Entwurfsmustern zählen zu den Problematiken der Studierenden.

Die dritte Kategorie beinhaltet alle Herausforderungen, die in Bezug auf die Syntax der UML auftreten können. Gerade die Vielzahl an Diagrammtypen mit teils ähnlich aussehenden Komponenten (beispielsweise Klassen und Aktivitäten), aber unterschiedlichen Bedeutungen ist zu Beginn eine Hürde für Studierende.

In der letzten Kategorie sind Herausforderungen beschrieben, die in Relation zur Bearbeitung textueller Anforderungsbeschreibungen stehen. Konkret genannt wurden die Extraktion aller beschriebenen Diagrammkomponenten sowie deren individuellen Beziehungen zueinander. Auch die beschränkte Verfügbarkeit von Aufgaben ist in einem Interview genannt worden. Alle Teilnehmenden haben die Bereitstellung zusätzlicher Aufgabenstellungen jedoch als Anforderung an eine softwaregestützte Anwendung gestellt, was die Wichtigkeit dieser Funktionalität unterstreicht.

3.3.7 Anforderungen von Studierenden an eine softwaregestützte Hilfestellung

Der Ablauf einer Übungseinheit lässt sich üblicherweise in drei Phasen unterteilen. Das Ergebnis der induktiven Kategoriebildung ist in Tabelle 3.14 einsehbar und spiegelt diese wieder. Nach der Bereitstellung und Bearbeitung der Aufgaben erfolgen Modellierung sowie ggf. ein Feedback (in der

ID	Herausforderung	Prozent
SIF1-K4-1	Extraktion von UML-Komponenten	80
SIF1-K4-2	Identifikation von Beziehungen der UML-Komponenten untereinander	60
SIF1-K4-3	Verfügbarkeit von textuellen Aufgaben	20

Tabelle 3.13: Qualitativ erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Bearbeitung textueller Aufgaben*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 191).

ID	Kategorie	Prozent
SIF2-K1	Bereitstellung und Bearbeitung textueller Aufgaben	100
SIF2-K2	Feedback und Verbesserungsvorschläge	100
SIF2-K3	Modellierung von UML-Diagrammen	40

Tabelle 3.14: Kategorien der qualitativ erhobenen Anforderungen von Studierenden an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 191).

Lehrpraxis meist vom Typ: Richtige Antwort) und Verbesserungsvorschläge. Die individuellen Anforderungen, die mithilfe des induktiven Vorgehens zu den genannten Kategorien führten, sind in den Tabellen 3.15, 3.16 und 3.17 zu sehen.

Die Ergebnisse verdeutlichen, dass Studierende sich besonders in der Phase der Bereitstellung und Bearbeitung textueller Aufgaben softwaregestützte Hilfestellungen wünschen und in dieser Kategorie auch die meisten Anforderungen erhoben werden konnten. Die Bereitstellung eines umfassenderen Übungsangebots ist in Unterabschnitt 3.3.6 beschrieben. Eine wichtige Erkenntnis von Huber u. a. (2023b) ist die Tatsache, dass Studierende für bereitgestellte, textuelle Anforderungsbeschreibungen auch mindestens eine zugehörige Musterlösung (vorzugsweise individuelles Feedback) fordern. Nur so ist für sie verifizierbar, ob ein erstelltes UML-Diagramm den Inhalten der Aufgabe vollständig entspricht. Neben der Menge an Übungsmaterial wurde auch die Schwierigkeit der Aufgabentexte thematisiert. Den unterschiedlichen Wissensständen der Studierenden mit unterschiedlichen Aufgaben zu

ID	Anforderung	Prozent
SIF2-K1-1	Größere Anzahl an textuellen Aufgaben	100
SIF2-K1-2	Markierung wichtiger Textbausteine, die auf UML-Komponenten hinweisen	100
SIF2-K1-3	Textuelle Aufgaben mit unterschiedlichen Schwierigkeitsgraden	60
SIF2-K1-4	Anzeige von Hilfstexten	40
SIF2-K1-5	Kollaborativer Arbeitsmodus	20
SIF2-K1-6	Guide, der schrittweise durch die Bearbeitung der Aufgabe führt	20
SIF2-K1-7	Bereitstellung unterschiedlicher Hilfestellungen bei verschiedenen Schwierigkeitsgraden	20
SIF2-K1-8	Strategie Guide	20
SIF2-K1-9	Umdrehen der Aufgabenstellung	20

Tabelle 3.15: Qualitativ erhobene Anforderungen von Studierenden an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Bereitstellung und Bearbeitung textueller Aufgaben*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 192).

begegnen ist während des Übungsbetriebs meist nicht möglich. Durch eine Software ist dies jedoch abbildbar. Softwaregestützte Hilfestellungen können den Anforderungen der Studierenden zufolge je nach Schwierigkeitsgrad aktiviert oder deaktiviert werden. Ein in allen Interviews genanntes Beispiel für diese ist die Markierung wichtiger Textbausteine, um die Identifikation der enthaltenen Komponenten zu erleichtern. Weiterhin kamen Anforderungen zur Anzeige von Hilfstexten und der Bereitstellung eines kollaborativen Arbeitsmodus auf. Für die Unterstützung in ihrer Vorgehensweise sind ein Strategie Guide sowie ein Ratgeber, der durch die schrittweise Bearbeitung der Aufgabe führt, genannt.

Die zweite Kategorie beinhaltet Anforderungen, die nach dem Modellierungsprozess gewünscht werden. In allen Interviews erwähnt wurde der Wunsch nach individuellem Feedback, gefolgt von der Bereitstellung einer Musterlösung. Wie die Aussagen der Teilnehmenden verdeutlichen, ist individuelles Feedback einer Musterlösung vorzuziehen. Letztere ist obsolet, sofern Ersteres vorhanden ist. Eine weitere genannte Anforderung war die Verlinkung weiterführender Literatur.

Überraschenderweise nannten die Teilnehmenden nur in 40% der In-

ID	Anforderung	Prozent
SIF2-K2-1	Individuelles Feedback	100
SIF2-K2-2	Bereitstellung einer Musterlösung	60
SIF2-K2-3	Bereitstellung weiterer Literaturvorschläge	20

Tabelle 3.16: Qualitativ erhobene Anforderungen von Studierenden an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Feedback und Verbesserungsvorschläge*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 192).

ID	Anforderung	Prozent
SIF2-K3-1	Verknüpfung des UML-Diagramms mit Programmcode	20
SIF2-K3-2	Bereitstellung eines UML-Editors	20
SIF2-K3-3	Erzeugung von Vorschlägen für den nächsten Schritt	20

Tabelle 3.17: Qualitativ erhobene Anforderungen von Studierenden an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Modellierung von UML-Diagrammen*. Übersetzt durch den Autor. Eigene Darstellung in Anlehnung an (Huber u. a., 2023b, S. 192).

terviews softwaregestützte Hilfestellungen, die während der Modellierung assistieren. Nach Aussagen der Teilnehmenden sei es für ihren Lernprozess wichtig, sich selbst und ohne Unterbrechungen an der Erstellung der UML-Diagramme zu versuchen. Zu diesen Anforderungen gehören die Verlinkung oder Visualisierung von Diagrammkomponenten mit zugehörigem Programmcode, ein simpler UML-Editor sowie Vorschläge für den nächsten Modellierungsschritt.

3.4 Qualitative Erhebung von Anforderungen der Dozierenden

Neben den im vorherigen Abschnitt erhobenen Anforderungen von Studierenden an das zu entwickelnde Artefakt, sind auch die Bedarfe von Dozierenden von entscheidender Bedeutung. Wie in den (Unter-)Abschnitten 2.1.3 und 3.2 aufgezeigt, ist die Wissensbasis in diesem Kontext ergiebiger. Die

darin beschriebenen Herausforderungen von Studierenden und Funktionalitäten softwaregestützter Hilfestellungen dienen als Entwicklungsgrundlage. Jedoch geben sie keinen hinreichenden Aufschluss darüber, wie ein ganzheitlicher Softwareprototyp aussehen müsste, um die relevantesten, in diesem Lehrkontext auftretenden Aspekte abzudecken und Studierende während des gesamten Prozesses zu unterstützen. Neben den Studierenden als Hauptakteure, sind auch die Ideen und Wünsche von Dozierenden eine essenzielle Grundlage in der Konzipierung und prototypischen Implementierung des Artefakts.

Eine quantitative Erhebung dieser Anforderungen, beispielsweise mithilfe eines Fragebogens, könnte, wie auch bei der Erhebung studentischer Anforderungen, zu einem ungenügenden Ergebnis führen. Eine qualitative Erhebung anhand eines leitfadengestützten Experteninterviews ist daher eine sinnvollere Wahl. In Anlehnung an Huber u. a. (2023b) orientieren sich Forschungsdesign und Vorgehen auch hier an Baur und Blasius (2019) sowie Bogner u. a. (2014).

3.4.1 Forschungsdesiderat und -fragen

Das zentrale Forschungsdesiderat des leitfadengestützten Experteninterviews ist die Anforderungserhebung von Dozierenden an eine softwaregestützte Hilfestellung für die Lehre von UML-Klassendiagrammen in einem hochschulischen Umfeld. Herausforderungen, die Lehrpersonen bei ihren Studierenden dabei erkennen, sind in der Literatur mehrfach untersucht worden. Auch erste Anforderungen sind anhand bestehender Lösungsansätze ableitbar (siehe Abschnitt 3.2). Das Forschungsinteresse dieser Untersuchung ist es, die vorhandene Wissensbasis zu erweitern und ein möglichst umfangreiches Portfolio an Herausforderungen sowie Anforderungen zu erhalten. Dies stellt sicher, dass die Konzeption des Artefakts möglichst viele relevante Aspekte berücksichtigt. Anforderungen von Dozierenden an eine softwaregestützte Hilfestellung können sich auf Funktionalitäten für Studierende oder Lehrpersonen beziehen. Das Forschungsinteresse inkludiert beide Aspekte.

Anhand des Forschungsdesiderats lassen sich drei zentrale Forschungsfragen ableiten:

- **DIF1:** Mit welchen Herausforderungen sehen sich die Studierenden eines Software Engieering-Kurses Ihrer Meinung nach konfrontiert, wenn es um das Erlernen der UML und die Erstellung zugehöriger Diagramme geht?
- **DIF2:** Welche softwaregestützten Hilfestellungen könnten Studierende von der Bereitstellung einer Aufgabe bis hin zu individuellem Feedback unterstützen?
- **DIF3:** Welche konkreten Funktionalitäten müssen für Sie als Lehrperson bereitgestellt werden, um eine Übungsstunde in diesem Kontext digital abzubilden?

Das Kürzel „DIF“ ist dabei frei gewählt und dient lediglich der eindeutigen Identifizierung der Forschungsfragen.

3.4.2 Struktur des qualitativen, leitfadengestützten Experteninterviews

Die Struktur des leitfadengestützten Experteninterviews orientiert sich an den beschriebenen Forschungsfragen. Der Interviewleitfaden ist nach den Vorgaben von Baur und Blasius (2019) sowie Bogner u. a. (2014) erstellt worden und findet sich in Anhang A.2. Dozierende gelten im Rahmen der Untersuchung als Experten, da sie sich in ihrer beruflichen Praxis regelmäßig mit dem beschriebenen Problemkontext auseinandersetzen und daher Informationen besitzen, die von anderen Personengruppen nicht erfasst werden können (vgl. Bogner u. a., 2014). Wie auch bei Huber u. a. (2023b) diente eine nicht kontextrelevante und einfach zu beantwortende Frage als Einstieg in das Gespräch:

- Wie viele Semester lehren Sie bereits Software Engineering?

Neben einem praktikablen Gesprächseinstieg zeigen die Antworten der Frage auf, dass die befragten Experten langjährige Erfahrung ausweisen und somit wichtige Beiträge zur Beantwortung der Forschungsfragen liefern können.

Anschließend können sich die Experten zu den Forschungsfragen äußern. Mit dem Ziel der Aufrechterhaltung des Gesprächsflusses, beinhaltet jede Frage einige Unterfragen (vgl. Huber u. a., 2023b). Diese müssen jedoch nicht zwangsläufig in jedem Gespräch gestellt werden, wenn das Augenmerk der Experten auf weiteren Aspekten liegt (vgl. Bogner u. a., 2014; vgl. Baur und Blasius, 2019).

3.4.3 Teilnehmende

Als mögliche Teilnehmende für das leitfadengestützte Experteninterview kommen Dozierende infrage, die Software Engineering und im speziellen UML-Diagramme in einem hochschulischen Umfeld lehren. Wichtig ist dabei, dass die befragten Personen einen informatiknahen Lehrstuhl an einer Hochschule oder Universität innehaben und mehrere Semester Erfahrung aufweisen. So wäre beispielsweise ein studentischer Übungsleiter in diesem Hinblick nicht als Experte zu bezeichnen.

Drei Dozierende unterschiedlicher deutscher Hochschulen und Universitäten haben sich zu einem Interview bereiterklärt. Die Befragungen fanden im Sommersemester 2023 statt. Da nach dem dritten Interview keine Erkenntnisse gewonnen werden konnten, die sich signifikant von denen aus den vorherigen Interviews unterschieden, ist davon auszugehen, dass die Anzahl der befragten Personen in diesem Kontext genügt.

3.4.4 Extraktion von Anforderungen

Die qualitative Inhaltsanalyse wird auch hier als intersubjektiv überprüfbare Auswertungsmethode (vgl. Mayring und Fenzl, 2022) herangezogen. Das Vorgehen orientiert sich dabei an den in Abbildung 3.2 dargestellten Schritten. Die von Huber u. a. (2023b) induktiv gebildeten Kategorien dienen hier als deduktive Kategorien, denen die Anforderungen von Dozierenden zugeordnet werden können. Eine Auflistung der deduktiven Kategorien für genannte Herausforderungen und Anforderungen finden sich in den Tabellen 3.18 und 3.19. Darin enthaltene IDs sind frei gewählt und dienen der eindeutigen Identifizierung angeführter Komponenten (dies gilt auch für alle weiteren Tabellen innerhalb des Abschnitts).

ID	Kategorie	Kodierregeln	Ankerbeispiel
DIF1-K1	Objekt-orientierung	Die getroffene Aussage nimmt konkreten Bezug auf Herausforderungen, die im Kontext der Objektorientierung auftreten.	„Zu erkennen, an welchen Stellen eine Vererbungsbeziehung zwischen zwei oder mehr Klassen notwendig ist, stellt für Studierende eine Herausforderung dar.“
DIF1-K2	Anwendung der UML	Die getroffene Aussage nimmt konkreten Bezug auf Herausforderungen, die im Kontext der Anwendung der UML auftreten.	„Die Vielzahl unterschiedlicher Möglichkeiten, einen Sachverhalt zu modellieren, stellt für Studierende eine Herausforderung dar.“
DIF1-K3	Komplexität der Syntax	Die getroffene Aussage nimmt konkreten Bezug auf Herausforderungen, die im Kontext der Komplexität und dem Verständnis der Syntax auftreten.	„Die Unterscheidung der zur Verfügung stehenden Beziehungskomponenten (beispielsweise Aggregationen und Kompositionen) stellt für Studierende eine Herausforderung dar.“
DIF1-K4	Bearbeitung textueller Aufgaben	Die getroffene Aussage nimmt konkreten Bezug auf Herausforderungen, die im Kontext der Bearbeitung textueller Aufgaben auftreten.	„Die Extraktion aller in einer textuellen Anforderungsbeschreibung enthaltenen Komponenten stellt für Studierende eine Herausforderung dar.“

Tabelle 3.18: Deduktiv erzeugte Kategorien für die qualitativ erhobenen Herausforderungen von Dozierenden, die sie bei Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen feststellen konnten. Die Darstellung orientiert sich an den Vorgaben von (Mayring und Brunner, 2006; Mayring, 2010).

ID	Kategorie	Kodierregeln	Ankerbeispiel
DIF23-K1	Bereitstellung und Bearbeitung textueller Aufgaben	Die getroffene Aussage nimmt konkreten Bezug auf Anforderungen an ein softwaregestütztes System, die sich auf die Bereitstellung und Bearbeitung textueller Aufgaben beziehen.	„Ich wünsche mir eine softwaregestützte Funktion, die Studierenden automatisiert textuelle Anforderungsbeschreibungen bereitstellt.“
DIF23-K2	Feedback und Verbesserungsvorschläge	Die getroffene Aussage nimmt konkreten Bezug auf Anforderungen an ein softwaregestütztes System, die sich auf die Bereitstellung von Feedback und Verbesserungsvorschlägen beziehen.	„Ich wünsche mir eine softwaregestützte Funktion, die Studierenden anhand des zuvor erzeugten Feedbacks Literaturvorschläge zur Verbesserung ihrer individuellen Leistung bereitstellt.“
DIF23-K3	Modellierung von UML-Diagrammen	Die getroffene Aussage nimmt konkreten Bezug auf Anforderungen an ein softwaregestütztes System, die sich auf die Modellierung von UML-Diagrammen beziehen.	„Ich wünsche mir eine softwaregestützte Funktion, die Studierenden einen simplen UML-Editor bereitstellt.“

Tabelle 3.19: Deduktiv erzeugte Kategorien für die qualitativ erhobenen Anforderungen von Dozierenden, an ein softwaregestütztes System für die Unterstützung von Studierenden in der hochschulischen Lehre von UML-Diagrammen. Die Darstellung orientiert sich an den Vorgaben von (Mayring und Brunner, 2006; Mayring, 2010).

ID	Herausforderung	Prozent
DIF1-K1-1	Abstraktheit der UML-Komponenten	67
DIF1-K1-2	Prinzip des Polymorphismus	67
DIF1-K1-3	Prinzip der Vererbung	67

Tabelle 3.20: Mithilfe qualitativer, leitfadengestützter Experteninterviews mit Dozierenden erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Objektorientierung*.

3.4.5 Übersicht über die Teilnehmenden

Drei Dozierende mit informatiknahen Lehrstühlen hatten sich für die Teilnahme an einem Interview bereiterklärt. Sie lehren an unterschiedlichen Hochschulen oder Universitäten und weisen zwischen 20 und 36 Semestern Erfahrung bei der Lehre von UML-Diagrammen und im speziellen von UML-Klassendiagrammen auf.

Aufgrund ihrer unterschiedlichen Standorte und der langjährigen Arbeit in den beschriebenen Fachgebieten ist davon auszugehen, dass vielfältige und fundierte Einblicke aus den Interviews gewonnen werden können.

3.4.6 Herausforderungen, die Dozierende bei ihren Studierenden bei der Modellierung von UML-Diagrammen sehen

Die in Tabelle 3.18 dargestellten deduktiven Kategorien dienen als Ausgangsbasis für die Einordnung der von Dozierenden im Rahmen der Interviews genannten Herausforderungen, die sie bei ihren Studierenden festgestellt haben. In den Tabellen 3.20, 3.21, 3.22 und 3.23 ist das Ergebnis dieser Zuordnung einsehbar. Die prozentualen Angaben beziehen sich auf die genannten Häufigkeiten einer Herausforderung (und Anforderungen, in den folgenden Unterabschnitten) über alle Interviews hinweg. Sie sind auf die nächstgelegene ganze Zahl gerundet.

Anhand bestehender Fachliteratur und den Ergebnissen der leitfadengestützten Experteninterviews mit Studierenden (siehe Unterabschnitt 3.3.6) ist ersichtlich, dass objektorientierte Konzepte für Studierende eine Herausforderung darstellen. Auch von den befragten Dozierenden wurde dieser

ID	Herausforderung	Prozent
DIF1-K2-1	Vielzahl unterschiedlicher Konzepte und Diagrammbestandteile	100
DIF1-K2-2	Verwendungszweck von Komponenten verstehen	100
DIF1-K2-3	Vielzahl der Modellierungsmöglichkeiten	100
DIF1-K2-4	Korrekte Verknüpfung der Komponenten untereinander	100
DIF1-K2-5	Beschriebene Konzepte mit den korrekten Komponenten modellieren	100
DIF1-K2-6	Komplexität von Modellierungstools	100
DIF1-K2-7	Zusammenhang zwischen UML-Diagrammen und Softwareentwicklungsphasen erkennen	33
DIF1-K2-8	Korrekte Auswahl und Anwendung von Entwurfsmustern	33
DIF1-K2-9	Grafische Darstellung theoretischer Konzepte	33

Tabelle 3.21: Mithilfe qualitativer, leitfadengestützter Experteninterviews mit Dozierenden erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Anwendung der UML*.

Aspekt angeführt und sie beschreiben, dass die Verknüpfung der zu erlernenden Konzepte mit bereits vorhandenem Wissen, gerade zu Beginn, für Lernende diffizil ist. Für viele sei die Modellierung von Sachverhalten durch ausgewählte Diagrammkomponenten im übertragenen Sinne nicht greifbar. Besonders deutlich wird dies laut den Dozierenden, wenn die Studierenden zusätzlich die Prinzipien des Polymorphismus oder der Vererbung berücksichtigen sollen.

Bei der praktischen Anwendung der UML wurde die Vielzahl unterschiedlicher Konzepte und Diagrammbestandteile von allen Dozierenden genannt. Zusätzlich scheint es den Studierenden schwer zu fallen, den Zusammenhang der vielen, unterschiedlichen Diagrammtypen und den dafür vorgesehenen Softwareentwicklungsphasen zu identifizieren. Demnach stellt nicht nur die reine Anzahl von Komponenten, sondern auch deren Einordnung sowie deren korrekte Auswahl bei der Modellierung von Sachverhalten eine Herausforderung dar. Häufig seien sich die Studierenden über den Verwendungszweck und die Einsatzmöglichkeiten von Diagrammbestandteilen

ID	Herausforderung	Prozent
DIF1-K3-1	Differenzierung der Komponenten	100
DIF1-K3-2	Unterschiedliches Vorwissen	33
DIF1-K3-3	Lesen und Verstehen von UML-Diagrammen	33

Tabelle 3.22: Mithilfe qualitativer, leitfadengestützter Experteninterviews mit Dozierenden erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Komplexität der Syntax*.

im Unklaren. Sofern sie diese Hürde überwunden haben, ist die Übersetzung der zu erlernten theoretischen Konzepte in eine grafische Darstellung eine weitere Schwierigkeit, die es zu überwinden gilt. Alle Experten sprachen an, dass Modellierungstools aufgrund ihres Funktionsumfangs Studierende in vielen Fällen überfordern und den Fokus oftmals vom Erlernen der UML auf das Erlernen der Software verschieben. Eine weitere Herausforderung sei die korrekte Auswahl und Anwendung von Entwurfsmustern zu gegebenen Fragestellungen. Studierende seien sich oft unsicher, für welchen Anwendungsfall der Einsatz eines spezifischen Musters sinnvoll ist. Die in den studentischen Experteninterviews bereits identifizierte Herausforderung der Vielzahl an Modellierungsmöglichkeiten für einen Sachverhalt (siehe Tabelle 3.11) wurde von allen Dozierenden genannt.

Bezogen auf die Komplexität der Syntax beschreiben die Dozierenden, dass die Unterscheidung von Diagrammbestandteilen (auch von unterschiedlichen Diagrammtypen) schwierig ist. Aufgrund unterschiedlicher Motivationen, Studiengänge, vorangegangene (Berufs-)Ausbildungen (und vieler weiterer Gründe) weisen Studierende unterschiedliche Wissensstände auf. Das Erlernen der Syntax weist daher individuelle Hürden auf. Den Interviewteilnehmenden zufolge ist auch das Lesen und Verstehen von UML-Diagrammen eine Schwierigkeit, mit der sich Lernende konfrontiert sehen.

Auch bei der Bearbeitung textueller Aufgaben identifizieren die Lehrpersonen unterschiedliche Herausforderungen. Zu nennen ist beispielsweise die Extraktion aller beschriebenen Diagrammbestandteile und die Zuordnung dieser zueinander. Sind Komponenten mithilfe von Synonymen in einem Text beschrieben, sind sich Studierende oft unschlüssig. Die befragten Dozierenden bestätigten, dass die Bereitstellung von individuellem und di-

ID	Herausforderung	Prozent
DIF1-K4-1	Extraktion von UML-Komponenten	100
DIF1-K4-2	Korrektheit eines eigenen Diagramms anhand einer Musterlösung feststellen	100
DIF1-K4-3	Keine Rückmeldung nach Bearbeitung	100
DIF1-K4-4	Verfügbarkeit textueller Aufgaben	67
DIF1-K4-5	Synonyme in Aufgabentexten	33
DIF1-K4-6	Zuordnung beschriebener Komponenten zu Klassen	33

Tabelle 3.23: Mithilfe qualitativer, leitfadengestützter Experteninterviews mit Dozierenden erhobene Herausforderungen von Studierenden bei der Modellierung von UML-Diagrammen anhand textueller Aufgabenstellungen für die Kategorie *Bearbeitung textueller Aufgaben*.

rekten Feedback aus zeitlichen Gründen nicht für alle Kursteilnehmenden realisierbar ist. In den meisten Fällen erhalten sie zwar eine Musterlösung, jedoch fiele es den Studierenden aufgrund der vielfältigen Modellierungsmöglichkeiten schwer zu identifizieren, ob es sich bei ihrem Diagramm um eine akzeptable Lösung handelt. Da sie jedoch häufig die einzige orientierungsgebende Rückmeldung für Studierende darstellt, sei deren Verfügbarkeit laut der Dozierenden dennoch relevant. Die beschränkte Anzahl von Übungsaufgaben ist eine weitere zu nennende Herausforderung.

3.4.7 Anforderungen an eine softwaregestützte Hilfestellung, die Dozierende für ihre Studierenden erheben

Dieser Unterabschnitt beinhaltet alle innerhalb der Experteninterviews mit Lehrenden genannten Anforderungen, die eine softwaregestützte Hilfestellung für Studierende bereitstellen sollte. Es erfolgt eine Einordnung dieser in die in Tabelle 3.19 definierten deduktiven Kategorien. Das Ergebnis ist in den Tabellen 3.24, 3.25, 3.26 dargestellt.

Genau wie die Studierenden (siehe Unterabschnitt 3.3.7) erhoben die befragten Dozierenden in der Phase der Bereitstellung und Bearbeitung textueller Aufgaben die meisten Anforderungen für Lernende. Konkret kamen Wünsche nach einer automatisierten Bereitstellung textueller Aufgaben mit unterschiedlichen Schwierigkeitsgraden und Themenschwerpunkten auf.

ID	Anforderung	Prozent
DIF23-K1-1	Größere Anzahl an textuellen Aufgaben	100
DIF23-K1-2	Textuelle Aufgaben mit unterschiedlichen Schwierigkeitsgraden	67
DIF23-K1-3	Schrittweise Heranführung an neue Themengebiete	67
DIF23-K1-4	Markierung wichtiger Textbausteine, die auf UML-Komponenten hinweisen	33
DIF23-K1-5	Abstraktionsfähigkeit verbessern	33
DIF23-K1-6	Aufzeigen unterschiedlicher Lösungsmöglichkeiten	33
DIF23-K1-7	Textuelle Aufgaben mit unterschiedlichen Themenschwerpunkten	33
DIF23-K1-8	Guide, der schrittweise durch die Bearbeitung der Aufgabe führt	33

Tabelle 3.24: Qualitativ erhobene Anforderungen von Dozierenden für Studierende an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Bereitstellung und Bearbeitung textueller Aufgaben*.

Auch das schrittweise Heranführen von Lernenden an neue Themengebiete wurde genannt. Weitere Hilfestellungen könnten das Markieren wichtiger Textbausteine, die Anzeige unterschiedlicher Lösungsmöglichkeiten sowie ein Guide, der schrittweise durch die Bearbeitung einer Aufgabe führt, darstellen. Weiterhin denkbar wäre auch eine Funktionalität, die Studierende dabei unterstützt ihre Abstraktionsfähigkeit zu verbessern.

Ein zentraler Punkt stellt zudem die automatisierte Bereitstellung von individuellem und direktem Feedback zu studentischen UML-Diagrammen dar. Darin berücksichtigt könnten Synonyme für die Bezeichnungen der

ID	Anforderung	Prozent
DIF23-K2-1	Individuelles Feedback	67
DIF23-K2-2	Detektion von Synonymen	33
DIF23-K2-3	Detektion, welche Textbausteine einer Klasse fälschlicherweise zugeordnet wurden	33

Tabelle 3.25: Qualitativ erhobene Anforderungen von Dozierenden für Studierende an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Feedback und Verbesserungsvorschläge*.

ID	Anforderung	Prozent
DIF23-K3-1	Import von UML-Diagrammen aus unterschiedlichen Modellierungstools	33
DIF23-K3-2	Bereitstellung von Tipps und Tricks	33

Tabelle 3.26: Qualitativ erhobene Anforderungen von Dozierenden für Studierende an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Modellierung von UML-Diagrammen*.

Klassen sein. Weiterhin wünschten die Interviewteilnehmenden eine Funktionalität, die erkennt, ob Studierende bestimmte Textbausteine zu einer falschen Klasse zugeordnet haben.

Bei der Modellierung von UML-Diagrammen kam die Anforderung nach einer Möglichkeit des Imports von UML-Diagrammen auf, die von unterschiedlichen Modellierungseditoren stammen. Auch die Bereitstellung von Tipps und Tricks könnte für Studierende eine wichtige Hilfestellung sein.

3.4.8 Anforderungen an eine softwaregestützte Hilfestellung, die Dozierende für Lehrpersonen erheben

Die Tabellen 3.27, 3.28, 3.29 beinhalten alle von Dozierenden genannten Anforderungen, die eine softwaregestützte Hilfestellung für Lehrpersonen bereitstellen sollte. Auffällig ist, dass die Anzahl gewünschter Funktionalitäten hier merklich geringer ist, als im vorherigen Unterabschnitt. Es lässt sich implizieren, dass der Fokus der befragten Dozierenden vermehrt auf der digitalen Unterstützung ihrer Studierenden lag. Eine Einordnung dieser erfolgte ebenfalls in die in Tabelle 3.19 beschriebenen deduktiven Kategorien. Die Nummerierung der Anforderungen ist daher fortlaufend.

Im Sinne der Digitalisierung der Lehre von UML-Diagrammen wurde die Bereitstellung einer Übungsaufgabe für eine Gruppe von Studierenden genannt. Diese müsse sowohl manuell, als auch automatisiert erstellbar sein. In letzterem Fall sollten in beliebiger Häufigkeit auch Entwurfsmuster in den textuellen Anforderungsbeschreibungen enthalten sein.

Nicht nur die Erstellung von Übungsaufgaben, sondern auch die Generie-

ID	Anforderung	Prozent
DIF23-K1-9	Bereitstellung einer Übungsaufgabe für eine Gruppe von Studierenden	33
DIF23-K1-10	Automatische Generierung textueller Aufgaben, die Entwurfsmuster enthalten	33

Tabelle 3.27: Qualitativ erhobene Anforderungen von Dozierenden für Lehrpersonen an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Bereitstellung und Bearbeitung textueller Aufgaben.*

ID	Anforderung	Prozent
DIF23-K2-4	Automatisierte Erzeugung von Musterlösungen zu Übungsaufgaben	67
DIF23-K2-5	Anzeigen einer Fehlerstatistik aller studentischen Abgaben	33

Tabelle 3.28: Qualitativ erhobene Anforderungen von Dozierenden für Lehrpersonen an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Feedback und Verbesserungsvorschläge.*

rung von Musterlösungen bewerten die Lehrpersonen als zeitintensiv. Eine softwaregestützte Hilfestellung zur automatisierten Erzeugung von Musterlösungen könnte hier Abhilfe schaffen. Auch das Anzeigen einer Fehlerstatistik von Studierenden, die auf Basis eines automatisierten Feedback für jedes eingereichte UML-Diagramm erstellt worden ist, sei für Dozierende eine interessante Funktionalität. Anhand dieser Informationen können sie ihre Kursinhalte am Fehlerprofil von Studierenden orientieren.

Als letzte Anforderung in diesem Unterabschnitt ist die Visualisierung von Musterlösungen anzuführen. Diese Funktionalität schafft die Möglichkeit, eine automatisiert erzeugte Lösung anzuzeigen und zu überprüfen, be-

ID	Anforderung	Prozent
DIF23-K3-3	Visualisierung von Musterlösungen	33

Tabelle 3.29: Qualitativ erhobene Anforderungen von Dozierenden für Lehrpersonen an eine Softwareanwendung zur Unterstützung der Lehre von UML-Diagrammen in der hochschulischen Lehre von Software Engineering für die Kategorie *Modellierung von UML-Diagrammen.*

vor studentische UML-Diagramme maschinell mit dieser verglichen werden.

3.5 Zusammenführung aller beschriebenen Anforderungen

In Unterabschnitt 2.1.3 finden sich in der Literatur beschriebene Herausforderungen für Studierende bei der Modellierung von UML-Diagrammen im Allgemeinen, speziell aber auch von UML-Klassendiagrammen. Eine Übersicht in der Literatur angeführten Softwareanwendung zur Unterstützung der hochschulischen Lehre von UML-Diagrammen ist in der SLR in Unterabschnitt 3.1 einsehbar. Eine Extraktion von Anforderungen der darin aufgelisteten Ansätze ist in Abschnitt 3.2 tabellarisch aufgelistet. Gemeinsam mit den in diesem Kapitel beschriebenen qualitativen Anforderungserhebungen mit Studierenden sowie Dozierenden entsteht eine umfassende Wissensbasis, anhand derer ein Prototyp für die in dieser Arbeit zu entwickelnde Softwareanwendung konzipierbar wird. Da die Forschungsfragen anhand der bestehenden Wissensbasis nicht beantwortbar waren, ist diese durch weitere Untersuchungen und einhergehenden Publikationen durch den Autor erweitert worden. Der von Hevner u. a. (2004) beschriebene Rigorositätszyklus ist somit mehrfach aktiv durchlaufen.

Die Anforderungen unterschiedlicher Herkunft sollen in diesem Kapitel gemeinsam aufgelistet und somit die singuläre Entwicklungsgrundlage für die folgenden Kapitel bilden. Abschnitt 3.2 gliedert die darin beschriebenen Ergebnisse nach Basisanforderungen sowie visuelle Anforderungen. Eine weitere Unterteilung erfolgt anhand der Hauptakteure des geplanten Prototyps:

- Studierende (siehe Tabelle 3.5 und 3.7).
- Dozierende (siehe Tabelle 3.4).
- System (siehe Tabelle 3.6 und 3.8).

Diese Unterteilung wird bei der Zusammenführung aller Anforderungen herangezogen und erweitert. Weiterhin soll der Prototyp zur Überwindung beschriebener Problematiken beitragen. Um diese implementierbar zu machen,

findet eine Umwandlung in konkrete Anforderungen statt. Das folgende Beispiel verdeutlicht diesen Vorgang:

> **Herausforderung (aus Tabelle 3.12 entnommen):**
> Differenzierung der Komponenten.
>
> **Überführung der Problemstellung in eine softwaregestützte Hilfestellung:**
> Studierende stehen vor der Herausforderung, viele unterschiedliche und teils ähnlich aussehende Diagrammkomponenten voneinander zu unterscheiden. Die Bereitstellung einer Auflistung einsetzbarer Diagrammkomponenten (in dieser Arbeit ausschließlich Komponenten von UML-Klassendiagrammen) kann eine Lösung darstellen.
>
> **Anforderung:**
> Das System sollte Studierenden die Möglichkeit bieten, alle einsetzbaren UML-Klassendiagrammkomponenten anhand einer Auflistung einzusehen.

Die Zusammenführung aller Anforderungen findet sich als Ergebnis dieses Kapitels in den Tabellen 3.30, 3.31, 3.32, 3.33, 3.34. Das Kürzel „A“ vor einer eindeutigen Nummer steht hierbei für „Anforderung“. Die Beschreibung erfolgte nach den Vorgaben von Rupp u. a. (2014) auf dem Spezifikationslevel 2. Jede Anforderung ist mit einer neu vergebenen ID eindeutig identifizierbar. Die Tabellen enthalten außerdem die ursprünglichen Identifikationsnummern, mit denen die Herausforderungen respektive Anforderungen in den vorangegangenen Kapiteln referenziert waren. Somit ist eindeutig nachvollziehbar, woher diese stammen und ggf. auf welche Literaturangaben diese verweisen. Ein Tabelleneintrag kann seinen Ursprung in unterschiedlichen Quellen haben. Einige Quellen sind bei mehreren Tabelleneinträgen referenziert, sofern es unterschiedlicher Funktionalitäten bedarf, um die ursprünglich beschriebene Herausforderung oder Anforderung zu erfüllen.

In den Tabellen nicht enthalten sind die in Abschnitt 2.1.3 angeführten Herausforderungen HLA1 bis HLA10, die generell bei der hochschulischen Lehre von Software Engineering auftreten und zu unspezifisch sind, um sich

im Kontext dieser Arbeit in konkrete Funktionalitäten überführen zu lassen.

Für die Implementierung von (unter anderem) Deep-Learning-Modellen bietet sich die Python-Programmiersprache aufgrund vieler zur Verfügung stehender open-source Bibliotheken an. Das Vorhaben der Arbeit ist durch diese Hilfestellungen besser / effizienter realisierbar als mit anderen Programmiersprachen. Daher findet Python in diesem Kontext Anwendung. Dabei handelt es sich um keine konkrete Anforderung, jedoch um eine Festlegung für die folgenden Kapitel.

Bei der Modellierungssoftware Enterprise Architect handelt es sich um eine etablierte Anwendung, die sowohl in der hochschulischen Lehre als auch in Unternehmen eingesetzt wird. Beispielhaft sollen automatisiert erstellte UML-Klassendiagramme darin importierbar sein. Zudem wird davon ausgegangen, dass Studierende sowie Lehrpersonen diese für die Erstellung von Musterlösungen verwenden. Eine Bereitstellung der Modellierungsumgebung innerhalb eines eigenen Softwareprototyps ist durch fehlende Schnittstellen nicht möglich. Daher soll der Prototyp eine open-source Lösung als Alternative bereitstellen. Der Fokus innerhalb dieser Arbeit liegt jedoch auf dem Enterprise Architect. Für den Austausch exportierter Modellierungsdateien stellt die Object Management Group (OMG) den sogenannten XML Metadata Interchange (XMI) Standard bereit (vgl. Object Management Group, 2015). Speicherbar sind diese mit der Dateiendung .xmi oder .xml. Die Struktur des Inhalts folgt dem soeben genannten Standard. Im Folgenden wird ausschließlich von XML-Dateien gesprochen. Es sei jedoch explizit darauf verwiesen, dass deren Inhalte dem XMI-Format entsprechen.

Nach Konvention erfolgt die Beschreibung von UML-Klassendiagrammkomponenten (ggf. innerhalb textueller Anforderungsbeschreibungen und immer innerhalb zugehöriger Musterlösungen) im Singular (vgl. Seidl u. a., 2015). Es sei jedoch explizit darauf verwiesen, dass alle Personengruppen

angesprochen sind, auch wenn personenbezogene Substantive aus dem genannten Grund im maskulinen Singular stehen.

ID	Anforderung	Quelle(n)
A1	Das System muss Studierenden die Möglichkeit bieten, individuelles und direktes Feedback zu erhalten. Die Anforderung existiert, um die Unterschiede zwischen einer studentischen Lösung und einer Musterlösung zu kommunizieren.	BSA9, HLK5, HLK7, SIF1-K2-1, SIF2-K2-1, DIF1-K2-4, DIF1-K2-5, DIF1-K4-2, DIF1-K4-3, DIF23-K1-6, DIF23-K2-1
A2	Das System muss Studierenden die Möglichkeit bieten, sich selbst automatisiert textuelle Aufgaben mit auf ihren Wissensstand angepassten Schwierigkeitsgraden zu generieren.	HLK8, HLK14, HLK22, SIF1-K2-5, SIF1-K4-3, SIF2-K1-1, SIF2-K1-3, DIF1-K1-2, DIF1-K1-3, DIF1-K4-4, DIF23-K1-1, DIF23-K1-2, DIF23-K1-5, BSYA10
A3	Das System muss Studierenden die Möglichkeit bieten, bei der automatisierten Erstellung textueller Aufgaben zwischen vorgegebenen Themenschwerpunkten zu wählen.	DIF23-K1-7, BSYA10
A4	Das System muss Studierenden die Möglichkeit bieten, automatisiert textuelle Aufgaben zu erstellen, zu deren Lösung ein Entwurfsmuster verwendet werden müssen.	DIF23-K1-10
A5	Das System muss Studierenden die Möglichkeit bieten, UML-Klassendiagramme mithilfe eines UML-Editors zu modellieren.	BSA3, HLK15, SIF1-K2-6, SIF2-K3-2, DIF1-K2-6, DIF1-K2-9

A6	Das System muss Studierenden die Möglichkeit bieten, mithilfe eines UML-Editors modellierte UML-Klassendiagramme zu speichern.	BSA3, HLK15, SIF1-K2-6, SIF2-K3-2, DIF1-K2-6, DIF1-K2-9
A7	Das System muss Studierenden die Möglichkeit bieten, kollaborativ an einer textuellen Anforderungsbeschreibung zu arbeiten.	SIF2-K1-5
A8	Das System muss Studierenden die Möglichkeit bieten, eine Musterlösung für eine angezeigte textuelle Aufgabe einzusehen.	SIF2-K2-2, DIF23-K3-3
A9	Falls die Studierenden bereits Feedback zu einem abgegebenen UML-Klassendiagramm erhalten haben, muss das System Studierenden die Möglichkeit bieten, Vorschläge zu weiterführender Literatur einzusehen.	SIF2-K2-3
A10	Das System muss Studierenden die Möglichkeit bieten, eine textuelle Anforderungsbeschreibung einzusehen.	BSA1
A11	Das System muss Studierenden die Möglichkeit bieten, die von Dozierenden bereitgestellte Information über das Lernziel einer textuellen Anforderungsbeschreibung einzusehen.	BSA2, DIF1-K2-7
A12	Das System muss Studierenden die Möglichkeit bieten, Fragen zu einer textuellen Anforderungsbeschreibung in einem Chat-Fenster zu stellen.	BSA4
A13	Das System sollte Studierenden die Möglichkeit bieten, Feedback für ein UML-Klassendiagramm während des Modellierungsvorgangs zu erhalten.	BSA5

A14	Das System sollte Studierenden die Möglichkeit bieten, textuelle Aufgaben zu vorhandenen UML-Klassendiagrammen zu schreiben.	SIF2-K1-9
A15	Das System sollte Studierenden die Möglichkeit bieten, UML-Klassendiagramme in XML-, JPG- oder PNG-Format hochzuladen.	BSA7
A16	Das System sollte Studierenden die Möglichkeit bieten, ein UML-Klassendiagramm an einen Server zu schicken.	BSA8
A17	Das System sollte Studierenden die Möglichkeit bieten, die während eines Vergleichs zweier UML-Klassendiagramme detektierten Differenzen einzusehen.	BSA9
A18	Nachdem die Studierenden Feedback durch das System erhalten haben, sollte das System Studierenden die Möglichkeit bieten, eine verbesserte Version ihres UML-Klassendiagramms hochzuladen.	BSA11

Tabelle 3.30: Zusammenführung der Basisanforderungen für Studierende.

ID	**Anforderung**	**Quelle(n)**
A19	Das System muss Dozierenden die Möglichkeit bieten, eine textuelle Anforderungsbeschreibung für die Lehre von UML-Klassendiagrammen einzugeben.	BDA1a
A20	Das System muss Dozierenden die Möglichkeit bieten, eine textuelle Anforderungsbeschreibung für die Lehre von UML-Klassendiagrammen zu speichern.	BDA1b

A21	Das System muss Dozierenden die Möglichkeit bieten, textuelle Informationen über das Lernziel einer zugehörigen textuellen Anforderungsbeschreibung einzugeben.	BDA2a
A22	Das System muss Dozierenden die Möglichkeit bieten, textuelle Informationen über das Lernziel einer zugehörigen textuellen Anforderungsbeschreibung zu speichern.	BDA2b
A23	Das System muss Dozierenden die Möglichkeit bieten, mehreren Studierenden zur gleichen Zeit eine textuelle Anforderungsbeschreibung bereitzustellen.	BDA3, DIF23-K1-9
A24	Das System muss Dozierenden die Möglichkeit bieten, eine Musterlösung in Form einer Bilddatei (im JPG- oder PNG-Format) oder einer XML-Datei für eine textuelle Anforderungsbeschreibung hochzuladen.	BDA4a, DIF23-K3-1
A25	Das System muss Dozierenden die Möglichkeit bieten, eine Musterlösung in Form einer Bilddatei (im JPG- oder PNG-Format) oder einer XML-Datei für eine textuelle Anforderungsbeschreibung zu speichern.	BDA4b, DIF23-K3-1
A26	Das System muss Dozierenden die Möglichkeit bieten, Musterlösungen anhand einer vorhandenen textuellen Anforderungsbeschreibung automatisiert generieren zu lassen.	DIF23-K2-4
A27	Das System sollte Dozierenden die Möglichkeit bieten, eine Musterlösung durch einen UML-Editor für eine textuelle Anforderungsbeschreibung zu erstellen.	BDA5a
A28	Das System sollte Dozierenden die Möglichkeit bieten, eine Musterlösung durch einen UML-Editor für eine textuelle Anforderungsbeschreibung zu speichern.	BDA5b

A29	Das System sollte Dozierenden die Möglichkeit bieten zu spezifizieren, welche Komponenten eines UML-Klassendiagramms das System bei einem Vergleich berücksichtigen soll.	BDA6
A30	Das System sollte Dozierenden die Möglichkeit bieten, Fragen von Studierenden zu einer textuellen Anforderungsbeschreibung in einem Chat-Fenster einzusehen.	BDA7
A31	Das System sollte Dozierenden die Möglichkeit bieten, Fragen von Studierenden zu einer textuellen Anforderungsbeschreibung in einem Chat-Fenster zu beantworten.	BDA8
A32	Das System sollte Dozierenden die Möglichkeit bieten, eine statistische Übersicht über die Anzahl an Fehlern in studentischen UML-Klassendiagrammen einzusehen.	BDA9
A33	Das System sollte Dozierenden die Möglichkeit bieten, die von Studierenden hochgeladenen UML-Klassendiagramme zu einer textuellen Anforderungsbeschreibung und deren Differenzen zu einer Musterlösung einzusehen.	BDA10
A34	Das System sollte Dozierenden die Möglichkeit bieten, Lehrinhalte in Form einer PDF-Datei hochzuladen.	VSYA7, DIF1-K2-5, DIF1-K2-7, DIF1-K2-8, DIF1-K3-1, DIF1-K3-2, DIF23-K1-8, SIF2-K1-6, SIF1-K3-1, SIF1-K3-2
A35	Das System sollte Dozierenden die Möglichkeit bieten, Lehrinhalte in Form einer PDF-Datei zu speichern.	VSYA7, DIF1-K2-5, DIF1-K2-7, DIF1-K2-8, DIF1-K3-1, DIF1-K3-2, DIF23-K1-8, SIF2-K1-6, SIF1-K3-1, SIF1-K3-2

Tabelle 3.31: Zusammenführung der Basisanforderungen für Dozierende.

ID	Anforderung	Quelle(n)
A36	Das System muss unterschiedliche Hilfsfunktionen je nach Schwierigkeitsgrad der textuellen Aufgabe aktivieren oder deaktivieren.	SIF2-K1-7
A37	Das System sollte studentische Aktivitäten während der Modellierung eines UML-Klassendiagramms loggen.	BSYA1
A38	Das System sollte die Möglichkeit besitzen, modellierte UML-Klassendiagramme in eine XML-Datei zu exportieren.	BSYA2
A39	Das System muss UML-Klassendiagramme in Form eines Bildes (JPG- oder PNG-Format) oder in Form einer XML-Datei importieren.	DIF23-K3-1
A40	Das System muss hochgeladene UML-Klassendiagramme in eine interne Struktur zu überführen, die Vergleiche zwischen mehreren UML-Klassendiagrammen erlaubt.	BSYA3
A41	Das System muss studentische UML-Klassendiagramme anhand eines Bewertungsalgorithmus benoten.	BSYA4
A42	Das System sollte Synonyme für Klassen- , Attribut- und Methodennamen in der Bewertung berücksichtigen.	BSYA5, DIF23-K2-2, BSYA12
A43	Nachdem zwei UML-Klassendiagramme miteinander verglichen wurden, sollte das System eine Liste mit weiterführender Literatur bereitstellen, die Studierende für die Verbesserung ihrer Leistung einsehen können.	BSA10, SIF2-K3-3
A44	Das System sollte Statistiken über die Anzahl detektierter Fehler in studentischen UML-Klassendiagrammen speichern.	BSYA6, DIF23-K2-5

A45	Das System muss textuelle Anforderungsbeschreibungen in eine Musterlösung im XML-Format umwandeln.	DIF23-K2-4, BSYA11
A46	Nach einem Vergleich zweier UML-Klassendiagramme muss das System Feedback bereitstellen.	BSYA7, DIF1-K4-2, DIF1-K4-3, DIF23-K2-1
A47	Das System muss zwei UML-Klassendiagramme miteinander vergleichen und deren Differenzen detektieren.	BSYA8
A48	Das System sollte studentische UML-Klassendiagramme mit einem vordefinierten Regelwerk vergleichen und Abweichungen detektieren.	BSYA9
A49	Das System sollte eine schrittweise Heranführung an UML-Klassendiagramme anzeigen.	DIF23-K1-3
A50	Das System sollte Textbausteine einer textuellen Anforderungsbeschreibung detektieren, die einer Klasse fälschlicherweise zugeordnet wurden.	DIF23-K2-3

Tabelle 3.32: Zusammenführung der Basis-Systemanforderungen.

ID	**Anforderung**	**Quelle(n)**
A51	Falls der Schwierigkeitsgrad einer textuellen Anforderungsbeschreibung einfach ist, muss das System Studierenden die Möglichkeit bieten, die in textuellen Anforderungsbeschreibungen enthaltenen Diagrammbestandteile anzuzeigen.	BSA6, HLK1, HLK3, HLK4, HLK6, HLK7, HLK9, HLK10, HLK11, HLK13, HLK16, HLK17, HLK18, HLK19, HLK20, HLK21, HLK22, HLK23, HLK24, HLK26, HLK28, HLK29, HLK30, SIF1-K1-1, SIF1-K1-2, SIF1-K1-3, SIF1-K2-2, SIF1-K3-1, SIF1-K4-1, SIF2-K1-2, SIF2-K1-6, DIF1-K1-1, DIF1-K2-3, DIF1-K3-2, DIF1-K3-3, DIF1-K4-1, SIF1-K4-2, DIF1-K4-6, DIF23-K1-4

A52	Das System muss Studierenden die Möglichkeit bieten, eine Übersicht der unterschiedlichen einsetzbaren UML-Klassendiagrammkomponenten einzusehen.	HLK1, HLK7, HLK9, HLK10, HLK11, HLK12, HLK26, SIF1-K1-1, SIF1-K1-2, SIF1-K1-3, SIF1-K2-2, SIF2-K1-4, SIF2-K1-6, SIF1-K3-2, DIF1-K3-1, DIF1-K1-1, DIF1-K2-1, DIF1-K2-2, DIF1-K2-3, DIF1-K2-5, DIF1-K3-2, DIF23-K1-4
A53	Das System muss Studierenden die Möglichkeit bieten, Beschreibungen zu den unterschiedlichen einsetzbaren UML-Klassendiagrammkomponenten einzusehen.	HLK2, HLK9, SIF1-K1-1, SIF1-K1-2, SIF1-K2-2, SIF1-K3-1, DIF1-K1-1, DIF1-K2-1, DIF1-K2-2, DIF1-K2-3, DIF1-K2-5, SIF1-K3-2, DIF1-K3-1, DIF1-K3-2, DIF1-K3-3, DIF1-K4-6
A54	Das System sollte Studierenden die Möglichkeit bieten, zusammengehörige UML-Klassendiagrammkomponenten durch Markierung zu identifizieren.	HLK25, HLK26, HLK27, SIF1-K1-1, SIF1-K1-2, SIF1-K1-3, SIF1-K2-2, SIF1-K4-2, SIF2-K1-2, SIF2-K1-6, DIF1-K1-1, DIF1-K2-3, SIF1-K4-2, DIF1-K4-5, DIF23-K1-4
A55	Das System muss Studierenden die Möglichkeit bieten, eine Strategiebeschreibung für die Modellierung von UML-Klassendiagrammen einzusehen.	SIF1-K2-3, SIF1-K2-4, SIF2-K1-6, SIF2-K1-8, DIF23-K1-6, DIF23-K1-8, DIF23-K3-2, VSYA7
A56	Falls die textuelle Aufgabe ein dem System bekanntes Entwurfsmuster beinhaltet, sollte das System Studierenden die Möglichkeit bieten, den Namen und eine Beschreibung des Entwurfsmusters einzusehen.	SIF1-K2-7, SIF1-K2-8, DIF1-K2-8
A57	Das System sollte Studierenden die Möglichkeit bieten, Programmcode für eine textuelle Anforderungsbeschreibung zu schreiben.	VSA1, SIF1-K1-1, SIF1-K1-2, SIF1-K1-4, SIF2-K3-1, DIF1-K1-1

A58	Das System sollte Studierenden die Möglichkeit bieten, Programmcode in transformierter Form als UML-Klassendiagramm einzusehen.	VSA2, SIF1-K1-1, SIF1-K1-2, SIF1-K1-3, SIF2-K3-1, DIF1-K1-1
A59	Das System sollte Studierenden die Möglichkeit bieten, automatisiert erzeugten Programmcode zu einem modellierten UML-Klassendiagramm einzusehen.	VSA3, SIF1-K1-1, SIF1-K1-2, SIF1-K1-3, SIF2-K3-1, DIF1-K1-1
A60	Das System sollte Studierenden die Möglichkeit bieten, den generierten Programmcode neben einem zugehörigen UML-Klassendiagramm anzuzeigen.	VSYA4
A61	Während der Modellierung mit einem bereitgestellten UML-Editor sollte das System Studierenden die Möglichkeit bieten, nach Schlüsselworten innerhalb des UML-Klassendiagramms zu suchen.	VSA4, DIF1-K2-6
A62	Das System sollte Studierenden die Möglichkeit bieten, ein 2D UML-Klassendiagramm in Augmented Reality Ansicht, 3D oder Virtual Reality Ansicht einzusehen.	VSA5
A63	Das System sollte Studierenden die Möglichkeit bieten, erstellten Programmcode auszuführen und die Veränderungen der Objektparameter zur Laufzeit anzuzeigen.	VSA6
A64	Das System sollte Studierenden die Möglichkeit bieten, mit erstellten UML-Klassendiagrammen in 2D, Augmented Reality, 3D oder Virtual Reality zu interagieren.	VSA7

Tabelle 3.33: Zusammenführung der Visualisierungsanforderungen für Studierende.

ID	Anforderung	Quelle(n)
A65	Falls Studierende ein 3D UML-Klassendiagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein 3D UML-Klassendiagramm zu konvertieren.	VSYA1
A66	Das System sollte verschiedene Informationsebenen (Klassen, Attribute, Methoden) einblenden.	VSYA2
A67	Falls Studierende ein Augmented Reality Diagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein Augmented Reality Diagramm zu konvertieren.	VSYA3
A68	Falls Studierende ein Virtual Reality UML-Klassendiagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein Virtual Reality UML-Klassendiagramm zu konvertieren.	VSYA5
A69	Falls Studierende im Programmcode eine Zeile markieren, sollte das System zugehörige Diagrammbestandteile in dem zugehörigen UML-Klassendiagramm hervorheben.	VSYA6

Tabelle 3.34: Zusammenführung der Visualisierungsanforderungen für das System.

KAPITEL 4

Generierung textueller Übungsaufgaben und zugehöriger Musterlösungen

Häufig bieten Universitäten und Hochschulen für angewandte Wissenschaften Informatikstudiengänge mit unterschiedlichen Schwerpunkten an, beispielsweise Game Engineering oder Wirtschaftsinformatik. Für Dozierende ist es eine herausfordernde Aufgabe, ihre (Software Engineering-) Studierenden zu motivieren und zu einer aktiven Teilnahme zu bewegen (vgl. Jurgelaitis u. a., 2019). Eine höhere Beteiligung und Begeisterung können durch kontextuell relevante Übungsaufgaben erzielt werden, da es den Studierenden dadurch leichter fällt, sich mit der Aufgabenstellung zu identifizieren. Das hat jedoch zur Folge, dass Dozierende meist unterschiedliche Aufgabenstellungen für die Software Engineering-Kurse der individuellen Studiengänge bereitstellen. Gerade für das Erlernen von Modellieren mithilfe der UML ist ein großes Übungsangebot entscheidend. Da es keine standardisierte Vorgehensweise bei der Erstellung gibt, müssen die Konzepte durch Übung und Wiederholung gefestigt werden (vgl. Kurkovsky, 2015). Die Lernstrategien sowie -geschwindigkeiten der Studierenden sind sehr unterschiedlich. Auch in der Literatur wird darauf hingewiesen, dass komplexe und lange Anforderungsbeschreibungen oftmals einen zu fordernden Charakter aufweisen und von Studierenden nicht oder nur teilweise korrekt gelöst werden können (vgl. Wei u. a., 2016). Folglich wäre es sinnvoll, Studierenden Aufgaben

mit unterschiedlichen Schwierigkeitsgraden anzubieten, die ihrem jeweiligen Wissensstand entsprechen. Die Bereitstellung eines solch umfassenden Übungsangebots ist für Dozierende in der Lehrpraxis aus zeitlichen Gründen nicht abbildbar (vgl. Huber und Hagel, 2022a).

Da mehrere korrekte Modellierungen für eine textuelle Anforderungsbeschreibung existieren können, fällt es den Studierenden schwer, ihre eigenen Leistungen zu evaluieren (vgl. Stikkolorum u. a., 2019). Daher ist eine zugehörige Musterlösung notwendig. Auch deren Erstellung ist ein zeitlicher Faktor für Dozierende.

Ein wichtiges Forschungsdesiderat (siehe Forschungsfrage 1.1 in Abschnitt 1.1) stellt somit die Konzeption und prototypische Implementierung eines Systems zur automatisierten Generierung kontextuell relevanter Aufgabentexte mit unterschiedlichen Schwierigkeitsgraden sowie zugehörigen Musterlösungen dar. Das System soll die Bereitstellung eines umfassenden Übungsangebots von Aufgaben zu UML-Klassendiagrammen für Software Engineering-Studierende ermöglichen. Darüber hinaus hätten sie, unabhängig von Dozierenden, Zugriff auf ein breites Spektrum an Übungsaufgaben, mit denen sie sich auf die Prüfung vorbereiten können.

4.1 Vorgehen und Einordnung in das Forschungsdesign

Im Sinne des DSR-Forschungsdesigns nach Hevner u. a. (2004) identifiziert der vorherige Abschnitt realweltliche Probleme, die in der Literatur, aber auch in der Lehrpraxis durch den Autor in der Umgebung (siehe Abbildung 1.2) beobachtet wurden. Die sich daraus ergebenden Anforderungen (siehe Kapitel 3) dienen als Grundlage für das zu entwickelnde Artefakt. Die Wissensbasis mit ihren Grundlagen sowie Methodiken ist zu Teilen in den Kapiteln 2 und 3 bereits erläutert. In Unterabschnitt 4.2 soll konkret erörtert werden, welche verwandten Arbeiten zu Forschungsfrage 1.1 existieren. Die Wissensbasis dient als anwendbare Grundlage für die Konzeption und prototypische Implementierung des Artefakts. Diese Inhalte finden sich jeweils in den Abschnitten 4.3 und 4.4 wieder. Aus Gründen der Übersichtlichkeit werden die Generierung textueller Aufgaben und die Erstellung von

UML-Klassendiagrammen anhand von Texten gesondert konzipiert, implementiert und auch evaluiert. Für Letzteres werden geeignete Methodiken ausgewählt, um die Ergebnisse anhand der Umgebung zu bewerten. Daraus können sich Verbesserungspotenziale für weitere Entwicklungsstufen ergeben. Eine Zusammenführung beider Komponenten ist in Abschnitt 4.5 vorgesehen. In einem finalen Schritt erfolgt die Beantwortung der Forschungsfrage 1.1 (siehe Abschnitt 4.6).

4.2 Verwandte Arbeiten (Related Work)

Die automatisierte Erstellung, Verarbeitung und Analyse von Texten sowie Audioinhalten in natürlicher Sprache mithilfe von Deep-Learning-Modellen erfreut sich in den vergangenen Jahren großer Beliebtheit und somit eines großen Interesses vonseiten der Forschungsgemeinschaft (vgl. Jang u. a., 2020; vgl. Iqbal und Qureshi, 2020). Die Anwendungsmöglichkeiten solcher Technologien in akademischen wie unternehmerischen Kontexten sind vielfältig (vgl. Khurana u. a., 2023).

Laut Huber und Hagel (2022a) besteht im akademischen Bereich ein großes Interesse an der automatisierten Bereitstellung themenspezifischer Übungsaufgaben für Studierende. Solche Ansätze entlasten Lehrpersonen und ermöglichen Lernenden den Zugriff auf ein großes Aufgabenspektrum (vgl. Huber und Hagel, 2022a). Wie in Kapitel 3 mehrfach durch Literaturrecherchen und Anforderungserhebungen mit beteiligten Akteuren festgestellt, würden Studierende von der Verfügbarkeit eines größeren Übungsangebots profitieren. Zum Zeitpunkt der Veröffentlichung konnte die bestehende Wissensbasis den Autoren Huber und Hagel (2022a) keine hinreichende Antwort auf die Frage bieten, wie für die hochschulische Lehre von UML-Klassendiagrammen adäquate textuelle Anforderungsbeschreibungen mit zugehörigen Aufgabenstellungen in deutscher Sprache automatisiert generiert werden können.

Es konnte eine weitere Softwarelösung identifiziert werden, die textuelle Anforderungsbeschreibungen in deutscher Sprache generieren kann. Dabei handelt es sich um das im Dezember 2022 veröffentlichte ChatGPT (vgl. OpenAI, 2023; vgl. Liu u. a., 2023). Der Nachfolger des sogenannten InstructGPT liefert bemerkenswerte Ergebnisse bei unterschiedlichen NLP-

Aufgaben (vgl. Ouyang u. a., 2022; vgl. Liu u. a., 2023). Aufgrund dieser beeindruckenden Performance erntete ChatGPT international große Anerkennung aber auch Kritik, beispielsweise im Kontext ethischer Fragestellungen (vgl. Liu u. a., 2023). Eine kostenfreie Version ist im Internet einsehbar[16]. Für weiterführende Anwendung (beispielsweise die Einbindung in eigene Softwareanwendungen) fallen Lizenzkosten an[17]. Ein durch den Autor durchgeführter Test mit der im Internet kostenfrei verfügbaren Variante ergab, dass ChatGPT grundlegend in der Lage ist, textuelle Anforderungsbeschreibungen sowie zugehörige Aufgabenstellungen für UML-Klassendiagramme automatisiert zu generieren. Eine Erstellung solcher Übungsaufgaben mit unterschiedlichen Schwierigkeitsgraden und hochschulischen Themenschwerpunkten (beispielsweise Wirtschafts- oder Gesundheitsinformatik) war in dieser Untersuchung nicht möglich. Als Eingabe erhielt das Modell Aufforderungen, entsprechende Übungsaufgaben für einen Themenschwerpunkt mit einem festgelegten Schwierigkeitsgrad zu erzeugen. ChatGPT lieferte unterschiedliche Texte in deutscher Sprache. Diese waren syntaktisch korrekt und semantisch sowohl sinnvoll als auch zusammenhängend. Nach mehrfacher Präzisierung der Aufforderung enthielten die Resultate textuelle Anforderungsbeschreibungen, wie sie im Kontext der Arbeit einsetzbar wären. Jedoch war trotz expliziter Angabe der Rahmenparameter bei unterschiedlichen erzeugten Übungsaufgaben keinerlei Steigerung des Schwierigkeitsgrades (beispielsweise von einfach zu schwer) und keine klare Abgrenzung hochschulischer Themenschwerpunkte erkennbar. Im Kontext dieser Arbeit respektive speziell im Hinblick auf die in Kapitel 3 beschriebenen Anforderungen ist daher davon auszugehen, dass ChatGPT in der aktuell zur Verfügung stehenden Version nicht vollumfänglich für die hochschulische Lehre von UML-Klassendiagrammen geeignet ist.

Die Ergebnisse der leitfadengestützten Experteninterviews verdeutlichen, dass direktes und individuelles Feedback zu studentischen UML-Klassendiagrammen gegenüber einer verfügbaren Musterlösung zu präferieren ist. Um dieses computergestützt bereitstellen zu können, ist eine interne Repräsentation der in der textuellen Anforderungsbeschreibung enthaltenen

[16] Verfügbar unter: https://chat.openai.com/
[17] Einsehbar unter: https://openai.com/pricing

Diagrammbestandteile vonnöten. Nur so ist die Korrektheit einer studentischen Lösung verifizierbar. Die automatisierte Erstellung einer Musterlösung (als Vergleichsobjekt) ist daher ein weiterer, relevanter Bestandteil dieses Kapitels. Auch in diesem Bereich existieren bereits unterschiedliche Softwareanwendungen und Publikationen, die sowohl in akademischen als auch in industriellen Zusammenhängen Einsatz finden.

Abdelnabi u. a. (2021) liefern eine Übersicht solcher Anwendungen, die in der Literatur beschrieben sind. Darin enthalten ist eine prägnante Beschreibung sowie eine Übersichtstabelle, welche die jeweiligen Ansätze miteinander vergleicht. Die Erhebung verdeutlicht, dass sich die beschriebenen Applikationen teils auf unterschiedliche Diagrammtypen fokussieren, häufig keine Texte in natürlicher Sprache entgegennehmen und oftmals nicht vollautomatisch arbeiten (vgl. Abdelnabi u. a., 2021). Unter anderem verwenden diese Ansätze NLP und / oder Heuristiken für ihre Auswertungen. Aufgrund linguistischer Differenzen sind diese jedoch häufig nicht auf Texte in natürlicher deutscher Sprache anwendbar und ermöglichen somit keinen Einsatz innerhalb deutschsprachiger Hochschulkurse.

Im Rahmen eines SLR führten Ahmed u. a. (2022) eine vergleichbare Untersuchung durch. Laut der Autoren ist keiner der von ihnen angeführten Ansätze für eine automatisierte Generierung von UML-Klassendiagrammen anhand deutschsprachiger Texte einsetzbar (vgl. Ahmed u. a., 2022). Daher sind auch diese im Rahmen der hochschulischen Lehre im deutschsprachigen Raum nicht anwendbar.

Auch in diesem Teilbereich ist ChatGPT anzuführen. Mithilfe des Modells ist es grundlegend möglich, Musterlösungen anhand textueller Anforderungsbeschreibungen zu erzeugen. Ein Test durch den Autor zeigte jedoch, dass diese nicht immer vollständig sind und teils in englischer Sprache sowie unterschiedlichen Formaten beziehungsweise Darstellungsweisen ausgegeben werden. Sofern ChatGPT Beziehungen zwischen den beschriebenen Klassen ausgab, was ohne Nachvollziehbarkeit der Gründe nur sporadisch geschah, handelte es sich dabei um ungerichtete oder in beide Richtungen gerichtete Assoziationen. Weitere relevante Beziehungstypen wie beispielsweise die Vererbungsbeziehung erkannte das Modell nicht. Daher ist auch hier anzunehmen, dass ChatGPT für den Einsatz in der hochschulischen Lehre von UML-Klassendiagrammen noch nicht vollumfänglich geeignet und

daher eine alternative Lösung vonnöten ist.

Eine Überführung und Speicherung von Informationen durch oder mithilfe von Machine- sowie Deep-Learning-Modellen ist aus wissenschaftlicher Sicht kein Novum. Hierfür gibt es eine Vielzahl von Ansätzen zur Repräsentation von Sprache (beispielsweise über Wortvektoren), Bildern etc. Eine Überführung deutschsprachiger textueller Anforderungsbeschreibungen in ein UML-Klassendiagramm lässt sich mit den vorhandenen Ansätzen jedoch nicht abbilden.

Sowohl für die automatisierte Erstellung textueller Anforderungsbeschreibungen als auch für die computergestützte Generierung von UML-Klassendiagrammen anhand dieser konnten keine Softwareanwendungen oder Publikationen identifiziert werden, die sich für den Einsatz in der hochschulischen Lehre im deutschsprachigen Raum eignen und somit die in Kapitel 3 erhobenen Anforderungen erfüllen würden. Weiterhin war es dem Autor nicht möglich Ansätze zu identifizieren, die beide Paradigmen miteinander vereinen.

Eine von Eigler u. a. (2023) durchgeführte systematische Literaturübersicht beschreibt Softwarelösungen für die Unterstützung der Lehre von Entwurfs- und Architekturmustern in einem hochschulischen Kontext. Eine Kombination dieser mit den zuvor beschriebenen zwei Paradigmen konnte jedoch nicht identifiziert werden (vgl. Eigler u. a., 2023) und soll daher im Folgenden Berücksichtigung finden.

4.3 Generierung textueller Übungsaufgaben

In diesem Abschnitt erfolgt eine Untersuchung über die automatisierte Generierung kontextuell relevanter Übungsaufgaben. Zu Beginn erfolgt die Erläuterung einer durchgeführten Vorarbeit. Diese dient als Grundlage für die Konzeption sowie prototypische Implementierung eines Softwareartefakts. Letztlich ist eine Evaluation vorgesehen. Das zu konzipierende Softwareartefakt trägt zur teilweisen Beantwortung von Forschungsfrage 1.1 und folglich der Erreichung der wissenschaftlichen Zielsetzung der Arbeit bei.

4.3.1 Vorarbeit

Die Vorarbeit gliedert sich in eine Beschreibung des Aufbaus der Studie sowie einer Präsentation der Resultate.

4.3.1.1 Aufbau der Studie

Durch den Autor dieser Arbeit (neben Weiteren) wurde bereits eine erste Untersuchung zu dieser Themenstellung veröffentlicht (vgl. Huber und Hagel, 2022a). Wie bereits beschrieben, ist eine einfache Bereitstellung von beliebigen Texten für diesen Lehrkontext nicht sinnvoll. Auch die Verwendung einer Schablone liefert keine hinreichenden Ergebnisse. Hierbei würden beispielsweise Substantive oder Verben ausgetauscht. Dadurch verändern sich die Namen der UML-Diagrammkomponenten, die zugrundeliegende Struktur des Diagramms bleibt jedoch bestehen (vgl. Huber und Hagel, 2022a). Nach einmaliger Bearbeitung einer solchen Aufgabe, könnten Studierende die bestehende Lösung heranziehen und darin lediglich Substantive und Verben mit denen des neuen Aufgabentexts tauschen.

Große Erfolge im Bereich des NLP konnten in den vergangenen Jahren durch Deep-Learning-Technologien erzielt werden (vgl. Jang u. a., 2020, vgl. Iqbal und Qureshi, 2020). Daher wählten auch Huber und Hagel (2022a) ein solches Modell. Die Autoren beschreiben die folgenden Anforderungen an Syntax und Semantik der zu erstellenden Texte (in Anlehnung an (vgl. Huber und Hagel, 2022a, S. 52). Übersetzung durch den Autor):

- Alle Sätze müssen leserlich und verständlich geschrieben sein.
- Es dürfen keine Rechtschreib- und Grammatikfehler enthalten sein.
- Es darf keine zusammenhangslosen Worte oder Wortgruppen geben.
- Alle Sätze müssen einen zusammenhängenden Text ergeben.

Zur Erreichung dieser Ziele setzen sie auf das in Unterabschnitt 2.2.4.2 beschriebene GPT-2. Die in der Publikation verwendete GPT-2 Simple[18] Variante ist ohne spezifische Hardware auf eigene Anwendungsfälle erweiterbar. Dafür haben Huber und Hagel (2022a) 200 unterschiedliche Aufgabentexte erstellt. Diese beinhalten Beschreibungen von mystischen Kreaturen,

[18]Beschreibung und Code verfügbar unter: https://github.com/minimaxir/gpt-2-simple

die diverse Waffen und Items besitzen. Für jede Kreatur können Attribute und Methoden beschrieben sein. Auch Erklärungen von Entwurfsmustern sind in einigen der Texte enthalten. Die Übungsaufgabe spiegelt (nach den Erfahrungen des Autors an der Hochschule für angewandte Wissenschaften in Kempten, Deutschland) die Inhalte einer Übungseinheit für den Studiengang Informatik - Game Engineering wieder. Eine Eignung der Texte für diesen Anwendungsfall ist somit gegeben. Huber und Hagel (2022a) beschreiben keinerlei Differenzen bei der Komplexität ihrer Beispiele. Eine Komplexitätsmetrik für die Übungsaufgaben ist folglich in dieser Entwicklungsstufe nicht gegeben respektive ableitbar. Sie entstanden rein auf Basis des Erfahrungswissen der Autoren. Der Ansatz orientiert sich dabei konzeptionell an nur einem spezifischen Themenschwerpunkt.

Ein Trainingsbeispiel kann folgendermaßen aussehen (in Anlehnung an (Huber und Hagel, 2022a, S. 52). Übersetzung durch den Autor):

Ein Einhorn ist eine magische Kreatur und somit eine Landkreatur. Seine Waffe ist ein magisches Horn. Außerdem kann es fliegen und kennt seinen Namen. Die Wassernixe wiederum ist eine Wasserkreatur mit einem Dreizack, der Methode jagen und dem Attribut Alter. Treffen diese Kreaturen an Orten wie einem Strand aufeinander, kommt es zu einem Kampf. Eine historische Klasse zur Kalkulation von Schadenspunkten des Programms kann dabei nur mithilfe eines Adapters angesprochen werden.

Diese Trainingsbeispiele bieten sich beispielsweise für Game Engineering-Studierende an. Nach dem Training ist das Modell in der Lage, eigene und diverse Trainingstexte zu erzeugen. Die Publikation setzt die Programmiersprache Python ein.

Der gesamte Erstellungsprozess einer Übungsaufgabe ist von Huber und Hagel (2022a) in ein Modell mit vier Schritten untergliedert. Dieses ist in Abbildung 4.1 einsehbar. Zuallererst generiert das trainierte Modell einen Text. Dieser wird mit der folgenden Aufgabenstellung verknüpft (in Anlehnung an (Huber und Hagel, 2022a, S. 53). Übersetzung durch den Autor):

a) Beschreiben Sie die Klassen in UML-Notation mit sinnvollen Attributen und Methoden.

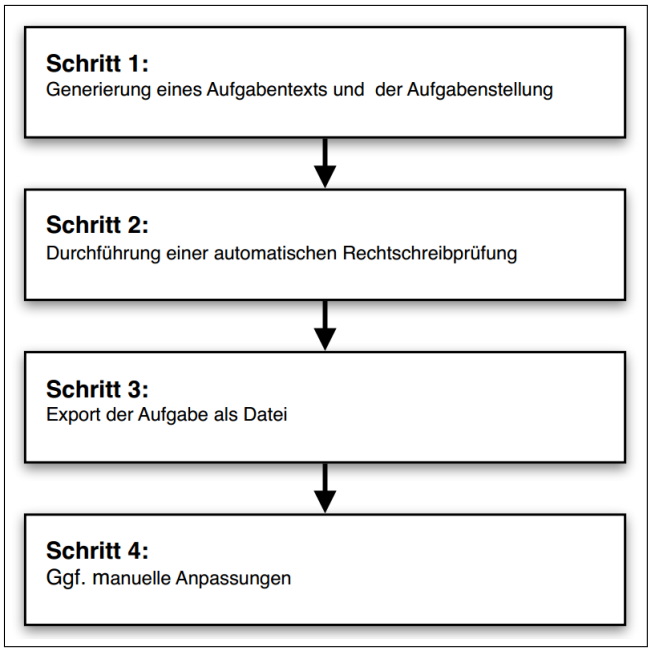


Abbildung 4.1: Vorgehen bei der Generierung textueller Aufgaben für die hochschulische Lehre von UML-Klassendiagrammen nach der Veröffentlichung von Huber und Hagel (2022a). Die Abbildung entstand in Anlehnung an (Huber und Hagel, 2022a, S. 53). Übersetzung durch den Autor.

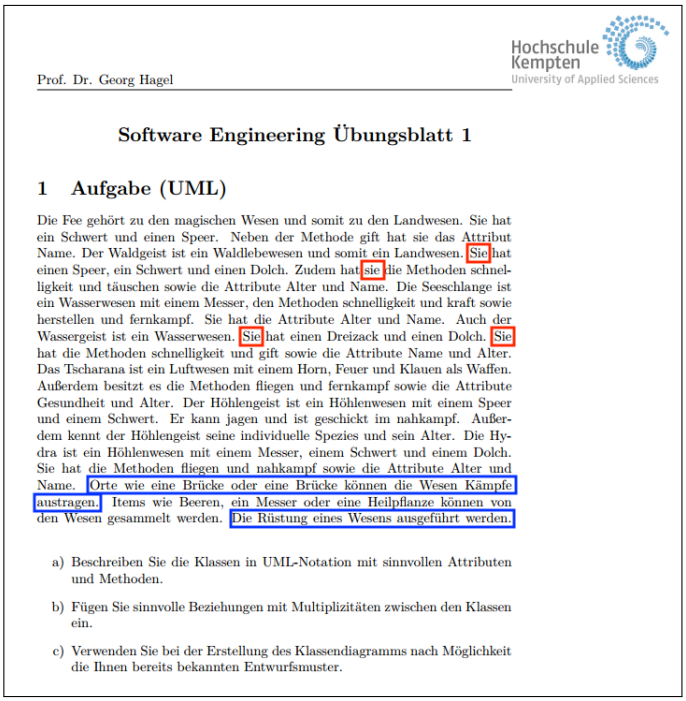
b) Fügen Sie sinnvolle Beziehungen zwischen diesen Klassen ein.

c) Wenn möglich, verwenden Sie die Ihnen bereits bekannten Entwurfsmuster bei der Erstellung Ihrer UML-Klassendiagramme.

Anschließend wird eine automatische Rechtschreibprüfung mithilfe einer Python-Bibliothek[19] durchgeführt. Der nächste Schritt beinhaltet die Generierung eines LaTeX-Dokuments mit zugehörigem PDF. Auch hierfür kommt eine spezielle Python-Bibliothek[20] zum Einsatz. Zusätzliche, in dem Dokument enthaltene Rechtschreib- und Grammatikfehler lassen sich manuell innerhalb der LaTeX-Datei korrigieren. Der vorgestellte Ansatz ist demnach ein semi-automatisches Verfahren, da eine manuelle Verbesserung der Texte erforderlich sein kann (vgl. Huber und Hagel, 2022a).

Abbildung 4.2 stellt das Ergebnis dieser Prozedur exemplarisch dar. Die innerhalb des abgebildeten Aufgabentexts sichtbaren roten Markierungen weisen auf Grammatikfehler hin (vgl. Huber und Hagel, 2022a). In allen Fällen setzt das Modell ein weibliches Personalpronomen für einen Waldgeist

[19] Verfügbar unter: https://pypi.org/project/pyspellchecker/
[20] Verfügbar unter: https://pypi.org/project/PyLaTeX/



Prof. Dr. Georg Hagel

Hochschule Kempten
University of Applied Sciences

Software Engineering Übungsblatt 1

1 Aufgabe (UML)

Die Fee gehört zu den magischen Wesen und somit zu den Landwesen. Sie hat ein Schwert und einen Speer. Neben der Methode gift hat sie das Attribut Name. Der Waldgeist ist ein Waldlebewesen und somit ein Landwesen. Sie hat einen Speer, ein Schwert und einen Dolch. Zudem hat sie die Methoden schnelligkeit und täuschen sowie die Attribute Alter und Name. Die Seeschlange ist ein Wasserwesen mit einem Messer, den Methoden schnelligkeit und kraft sowie herstellen und fernkampf. Sie hat die Attribute Alter und Name. Auch der Wassergeist ist ein Wasserwesen. Sie hat einen Dreizack und einen Dolch. Sie hat die Methoden schnelligkeit und gift sowie die Attribute Name und Alter. Das Tscharana ist ein Luftwesen mit einem Horn, Feuer und Klauen als Waffen. Außerdem besitzt es die Methoden fliegen und fernkampf sowie die Attribute Gesundheit und Alter. Der Höhlengeist ist ein Höhlenwesen mit einem Speer und einem Schwert. Er kann jagen und ist geschickt im nahkampf. Außerdem kennt der Höhlengeist seine individuelle Spezies und sein Alter. Die Hydra ist ein Höhlenwesen mit einem Messer, einem Schwert und einem Dolch. Sie hat die Methoden fliegen und nahkampf sowie die Attribute Alter und Name. Orte wie eine Brücke oder eine Brücke können die Wesen Kämpfe austragen. Items wie Beeren, ein Messer oder eine Heilpflanze können von den Wesen gesammelt werden. Die Rüstung eines Wesens ausgeführt werden.

a) Beschreiben Sie die Klassen in UML-Notation mit sinnvollen Attributen und Methoden.

b) Fügen Sie sinnvolle Beziehungen mit Multiplizitäten zwischen den Klassen ein.

c) Verwenden Sie bei der Erstellung des Klassendiagramms nach Möglichkeit die Ihnen bereits bekannten Entwurfsmuster.

Abbildung 4.2: Beispiel einer generierten, textuellen Aufgabe. Die Abbildung wurde aus (Huber und Hagel, 2022a, S. 53) entnommen.

oder Wassergeist ein. Das Wort „Geist“ scheint mit der Verwendung dieses Personalpronomens assoziiert zu sein (vgl. Huber und Hagel, 2022a). Die blauen Markierungen weisen auf Sätze hin, die inhaltlich nicht sinnvoll sind (vgl. Huber und Hagel, 2022a). Das abgebildete Beispiel wurde von den Autoren gewählt, um die Schwächen des Modells aufzuzeigen.

Ein weiterer Fokus der Publikation liegt auf einer Evaluation des Modells mit Studierenden. Die Untersuchung soll offenlegen, ob die erstellten Aufgaben für den Einsatz in der hochschulischen Lehre von UML-Klassendiagrammen geeignet sind. Dabei geht es zum einen um Verständlichkeit sowie Nachvollziehbarkeit der Texte, als auch um die Angemessenheit des Schwierigkeitsgrades. Die in Abbildung 4.2 markierten Satzbestandteile wurden manuell ausgebessert und der Text anschließend für die Evaluation herangezogen.

Die an der deutschen Hochschule für angewandte Wissenschaften Kempten durchgeführte quantitative Untersuchung stellte 19 Studierenden eines Software Engineering-Kurses Lehrvideos zu UML-Klassendiagrammen und Entwurfsmustern bereit (vgl. Huber und Hagel, 2022a). Weiterhin erhielten die Studierenden die in Abbildung 4.2 dargestellte Aufgabe von den

Autoren. Nach Bearbeitung dieser bekamen sie einen Fragebogen. Dessen Aufbau gestalteten Huber und Hagel (2022a) anhand bestehender Fachliteratur (vgl. Porst, 2014; vgl. Döring und Bortz, 2016). Dieser beinhaltet sowohl offene, als auch geschlossene Fragen (vgl. Porst, 2014). Erstere können von den Studierenden frei beantwortet werden (beispielsweise über ein Freitextfeld). Für letztere gibt es (bis auf eine Ausnahme) die folgenden Antwortmöglichkeiten (in Anlehnung an (Huber und Hagel, 2022a, S. 53). Übersetzung durch den Autor):

- Ich stimme nicht zu.
- Ich stimme eher nicht zu.
- Ich stimme eher zu.
- Ich stimme voll zu.
- Kann ich nicht beurteilen.

Die Struktur des Fragebogens ist im Folgenden beschrieben (in Anlehnung an (vgl. Huber und Hagel, 2022a, S. 53f.). Übersetzung durch den Autor):

- Informationen über die partizipierenden Studierenden:
  - F1 Wie alt sind Sie?
  - F2 Welches Geschlecht haben Sie?
  - F3 Bitte nennen Sie Ihren Studiengang.
- Geschlossene Fragen zu der gestellten Übungsaufgabe:
  - F4 Den Aufgabentext habe ich, bezogen auf Grammatik und Satzbau, problemlos verstanden.
  - F5 Ich konnte voll und ganz nachvollziehen, was anhand der gegebenen Aufgabenstellung bearbeitet werden soll.
  - F6 Den Schwierigkeitsgrad der Aufgabenstellung empfinde ich als angemessen.
  - F7 Durch die Bearbeitung der Aufgabe habe ich etwas gelernt.

F8 Aus aktueller Sicht bin ich der Meinung, eine Aufgabe mit vergleichbarem Schwierigkeitsgrad in der Prüfung lösen zu können.

F9 Unter der Voraussetzung, dass ich Zugriff auf Aufgaben mit vergleichbarem Schwierigkeitsgrad hätte, würde ich diese für die Prüfungsvorbereitung nutzen.

- Geschlossene Frage für eine allgemeine Bewertung der Aufgabe durch die Studierenden. Hierbei ist eine Skala von 0 bis 10 gegeben, wobei 0 sehr schlecht und 10 sehr gut bedeutet.

  F10 Wie würden Sie die Aufgabe insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

- Offene Fragen zu der gestellten textuellen Aufgabe:

  F11 Was könnte Ihrer Meinung nach an den Aufgabentexten verbessert werden?

  F12 Was könnte Ihrer Meinung nach an der Aufgabenstellung verbessert werden?

  F13 Wie könnte eine computergestützte Anwendung zur Erstellung vergleichbarer Aufgabenstellungen in der Lehre eingesetzt werden?

  F14 Sonstiges Feedback.

4.3.1.2 Ergebnisse der Studie

Die Ergebnisse der von Huber und Hagel (2022a) durchgeführten Studie sind in diesem Unterabschnitt angeführt und beschrieben. Eine visuelle Aufbereitung der Antworten für die Fragen F4 - F9 sowie für die Frage F10 finden sich in Abbildung 4.3 sowie 4.4. Die prozentualen Angaben sind gerundet (vgl. Huber und Hagel, 2022a).

Beschreibung der Ergebnisse (in Anlehnung an (vgl. Huber und Hagel, 2022a, S. 54). Übersetzung durch den Autor):

- Demografische Informationen zu den partizipierenden Studierenden:

  F1 Wie alt sind Sie?
  Die befragten Studierenden waren zwischen 17 und 29 Jahre alt. Das durchschnittliche Alter beträgt 22.7 Jahre.

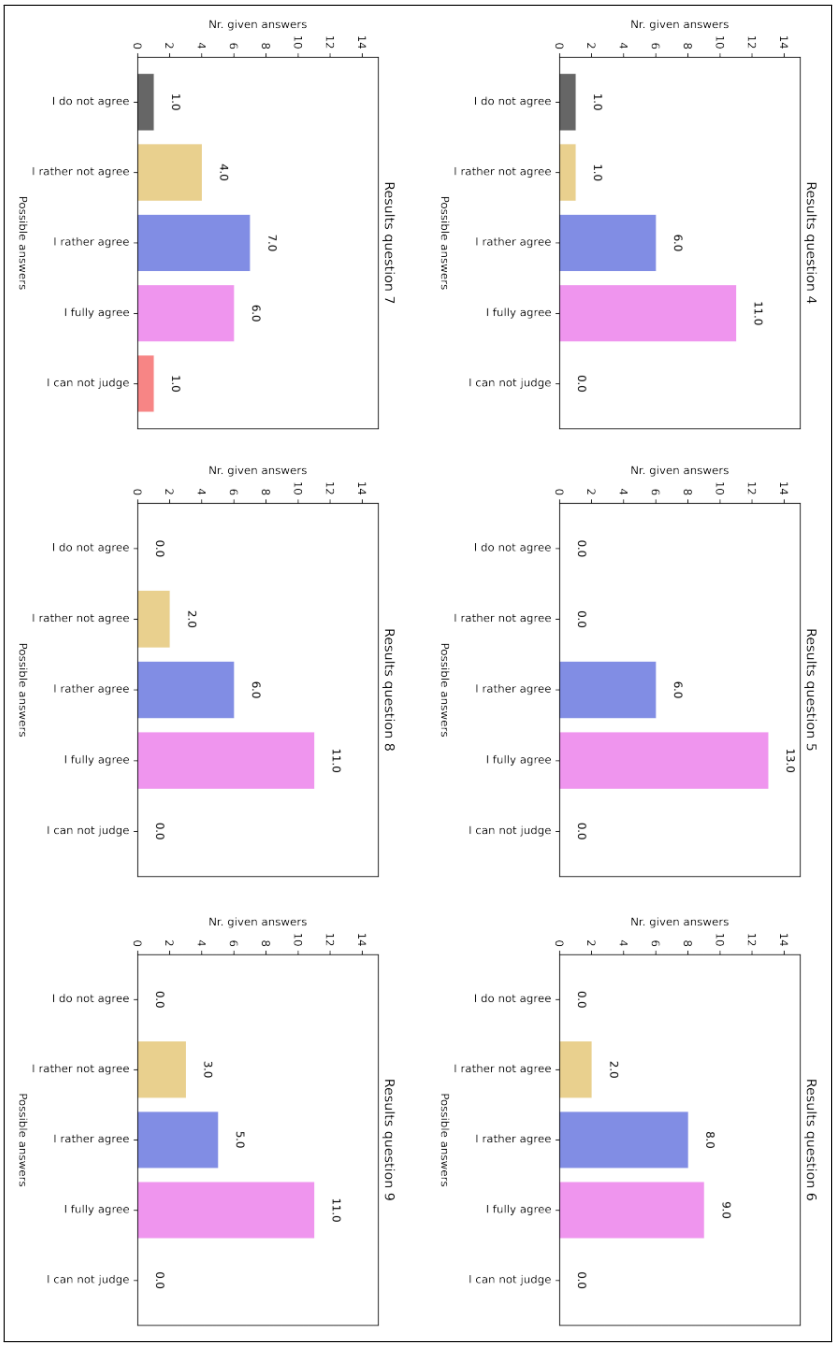


Abbildung 4.3: Darstellung der Ergebnisse für die Fragen F4 - F9 der von Huber und Hagel (2022a) durchgeführten quantitativen Studie zur Eignung eines Textgenerierungsmodells für die hochschulische Lehre von UML-Klassendiagrammen. Die Abbildung wurde aus (Huber und Hagel, 2022a, S. 55) entnommen.

F2 Welches Geschlecht haben Sie?
Fünf der Studierenden waren weiblich, 14 männlich. Unter den Teilnehmenden wählte niemand die Antwortmöglichkeiten „Divers“ oder „Keine Antwort“.

F3 Bitte nennen Sie Ihren Studiengang.
Elf Studierende gaben an, Informatik zu studieren. Acht gaben an, Informatik - Game Engineering zu studieren.

- Geschlossene Fragen zu der gestellten textuellen Aufgabe:

F4 Den Aufgabentext habe ich, bezogen auf Grammatik und Satzbau, problemlos verstanden.
Die Studierenden stimmten dieser Aussage mit 31,8% eher und zu 57,9% voll zu. Lediglich zwei Teilnehmende (10,5%) gaben an, den Text nicht vollständig verstanden zu haben. Das Ergebnis zeigt, dass die meisten Studierenden den generierten Aufgabentext bezogen auf Grammatik und Satzbau (problemlos) verstehen.

F5 Ich konnte voll und ganz nachvollziehen, was anhand der gegebenen Aufgabenstellung bearbeitet werden soll.
Die Studierenden konnten die Aufgabenstellung zu 31,6% eher und zu 68,4% vollständig verstehen sowie nachvollziehen.

F6 Den Schwierigkeitsgrad der Aufgabenstellung empfinde ich als angemessen.
Zwei Studierende (10,5%) empfanden den Schwierigkeitsgrad der Aufgabe als nicht angemessen. Acht Studierende (42,1%) stimmten der Aussage eher zu. Neun der Teilnehmenden (47,4%) bewerteten den Schwierigkeitsgrad als vollkommen angemessen.

F7 Durch die Bearbeitung der Aufgabe habe ich etwas gelernt.
Ein Studierender (5,3%) stimmte dieser Aussage nicht zu. Weitere vier (21%) stimmten eher nicht zu. Wohingegen sieben (36,8%) eher und 31,6% voll zustimmten. Die Antwortmöglichkeit „Kann ich nicht beurteilen“ wurde einmal gewählt. Das Ergebnis zeigt, dass 68,4% der Studierenden bei der Bearbeitung der Aufgabe etwas gelernt haben.

F8 Aus aktueller Sicht bin ich der Meinung, eine Aufgabe mit vergleichbarem Schwierigkeitsgrad in der Prüfung lösen zu können.
57,9% der Studierenden stimmten voll und 31,6% eher zu.

F9 Unter der Voraussetzung, dass ich Zugriff auf Aufgaben mit vergleichbarem Schwierigkeitsgrad hätte, würde ich diese für die Prüfungsvorbereitung nutzen.
84,2% der Teilnehmenden können sich (voll und eher) vorstellen eine vergleichbare Aufgabe für die Prüfungsvorbereitung zu nutzen. Lediglich drei Studierende stimmten dieser Aussage eher nicht zu.

- Geschlossene Frage für eine allgemeine Bewertung der Aufgabe durch die Studierenden. Hierbei ist eine Skala von 0 bis 10 gegeben, wobei 0 sehr schlecht und 10 sehr gut bedeutet.

  F10 Wie würden Sie die Aufgabe insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?
  63,2% der Studierenden gaben eine Wertung größer oder gleich 7 ab. Weitere 36,8% gaben eine Bewertung zwischen 3 und 6 ab.

- Offene Fragen zu der gestellten textuellen Aufgabe:

  F11 Was könnte Ihrer Meinung nach an den Aufgabentexten verbessert werden?

  - Paragraphen innerhalb des Aufgabentexts wären hinsichtlich der Lesbarkeit sinnvoll.
  - Gerade im Hinblick auf die Prüfung erschien der Aufgabentext vielen Studierenden zu lang.
  - Einmal wurde der Wunsch nach komplexeren Satzstrukturen geäußert. Darin soll es merklich schwieriger sein, die Diagrammkomponenten zu identifizieren.

  F12 Was könnte Ihrer Meinung nach an der Aufgabenstellung verbessert werden?
  Die Studierenden äußerten keine Änderungswünsche hinsichtlich der Aufgabenstellung.

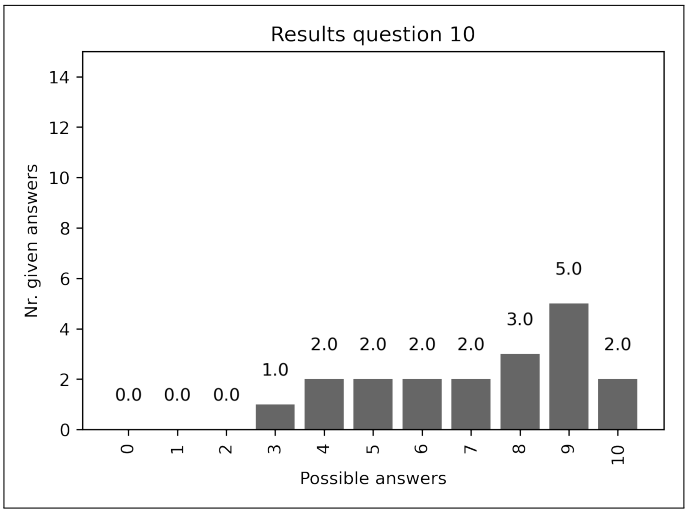


Abbildung 4.4: Darstellung des Ergebnisses für die Frage F10 der von Huber und Hagel (2022a) durchgeführten quantitativen Studie zur Eignung eines Textgenerierungsmodells für die hochschulische Lehre von UML-Klassendiagrammen. Die Abbildung wurde aus (Huber und Hagel, 2022a, S. 55) entnommen.

F13 Wie könnte eine computergestützte Anwendung zur Erstellung vergleichbarer Aufgabenstellungen in der Lehre eingesetzt werden?

Hauptsächlich antworteten die Studierenden, eine solche Anwendung zur Vorbereitung auf die Prüfung nutzen zu wollen. Neben der Aufgabe wäre es jedoch sehr hilfreich, auch eine Musterlösung bereitgestellt zu bekommen. Weiterhin kam der Wunsch nach Aufgabenstellungen mit unterschiedlichen Schwierigkeitsgraden auf.

F14 Sonstiges Feedback.

Es gab keinerlei weiteres Feedback, das für die Studie relevant gewesen wäre.

Die aufgelisteten Ergebnisse verdeutlichen, dass das entwickelte Modell zur semi-automatischen Erstellung von Aufgabentexten zur hochschulischen Lehre von UML-Klassendiagrammen von den Studierenden größtenteils positiv bewertet wurde. Der bereitgestellte Aufgabentext war, bezogen auf Satzbau und Grammatik, gut verständlich. Die damit einhergehende Aufgabenstellung konnten die Teilnehmenden gut nachvollziehen und es kamen keinerlei Änderungswünsche auf. Auch der Schwierigkeitsgrad wurde als an-

gemessen empfunden. Neben der Verständlichkeit von Text und Aufgabenstellung handelt es sich dabei um einen kritischen Punkt. Ziel dieser Übung ist es, Studierende ihrem Wissensstand entsprechend zu fordern, sie aber nicht zu überfordern. Neben einer angemessenen Komplexität der Aufgabe waren die Teilnehmenden überzeugt, bei der Bearbeitung etwas gelernt zu haben. Weiterhin sind sie der Ansicht, eine vergleichbare Aufgabe in einer Prüfungssituation lösen zu können. Stünde ihnen ein solcher Textgenerator zur Verfügung, würden sie diesen nach eigenen Angaben für die Vorbereitung auf eine Prüfung nutzen.

Das Gesamtergebnis der Studie ist positiv zu bewerten. Anhand der Änderungswünsche der Studierenden wird jedoch deutlich, dass das Modell noch einiges Verbesserungspotenzial aufweist.

Das von Huber und Hagel (2022a) entwickelte Artefakt wurde unter Zuhilfenahme der Wissensbasis entwickelt und in der Umgebung evaluiert, was wiederum zu einer Erweiterung der Wissensbasis führte (vgl. Hevner u. a., 2004). Die gleiche Vorgehensweise wird im Folgenden im Sinne des gewählten Forschungsdesigns verfolgt.

4.3.2 Konzeption

Im Sinne der von Hevner u. a. (2004) beschriebenen zyklischen Verfahrensweise (siehe Abbildung 1.1), greift die Konzeption in diesem Unterabschnitt den in der Vorarbeit erläuterten Ansatz auf und optimiert diesen entsprechend der in der Studie von Huber und Hagel (2022a) detektierten Wünsche sowie Vorstellungen der Studierenden ($\hat{=}$ Umgebung). Das resultierende Artefakt soll für sich allein nutzbar, aber auch in das zu konzipierende Gesamtsystem integrierbar sein.

Das in Abbildung 4.5 dargestellte UML-Paketdiagramm visualisiert den Aufbau der konzipierten Softwarekomponente. Eine vollständige Darstellung aller Attribute und Methoden ist aus Gründen der Übersichtlichkeit nicht möglich. Das Schaubild beinhaltet daher die relevantesten Informationen.

Übergeordnet folgt das Design dem Fassade-Entwurfsmuster (vgl. Eilebrecht und Starke, 2019; vgl. Kleuker, 2018). Zusammengehörige Klassen werden hierbei in ein gemeinsames Paket gruppiert und sind über ei-

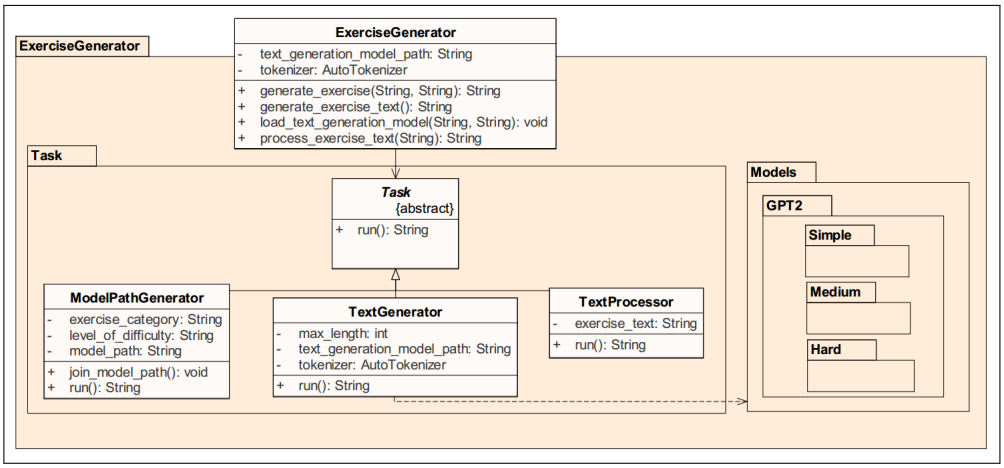


Abbildung 4.5: Softwaredesign der Softwarekomponente für die automatisierte Generierung textueller Anforderungs- und Aufgabenbeschreibungen.

ne sogenannte Fassadeklasse ansprechbar. Diese reguliert, in welcher Reihenfolge die gekapselten Klassen aufgerufen werden sollen und stellt weiterhin die Kommunikationsschnittstelle für externe Zugriffe (beispielsweise aus anderen Paketen) dar. Wie Abbildung 4.5 verdeutlicht, befinden sich alle hier beschriebenen Klassen innerhalb des Pakets ‚ExerciseGenerator'. Eine gleichnamige Klasse stellt die Fassadeklasse und somit die Kommunikationsschnittstelle nach außen bereit. In dem Unterpaket ‚Task' befindet sich die eigentliche Funktionalität der Softwarekomponente. Enthalten sind die Klassen ‚ModelPathGenerator', ‚TextGenerator' sowie ‚TextProcessor'. Diese erben von der abstrakten Klasse ‚Task' und besitzen alle eine ‚run()'-Methode. Die Aufgabenbereiche der einzelnen Task-Klassen sind nach dem sogenannten Separation of Concerns (im deutschen unter *Trennung bzw. Separierung der Zuständigkeiten* bekannt) Prinzip (vgl. Vogel u. a., 2011) voneinander separiert. Jeder Klasse (und auch deren Methoden) ist somit eine eindeutige Verantwortung (im englischen unter dem Begriff *Single Responsibility Principle* bekannt) zugewiesen (vgl. Martin, 2018). Die jeweiligen Klassen stellen folgende Funktionalitäten bereit:

- ‚ModelPathGenerator': Nimmt je eine Variable für einen Schwierigkeitsgrad und ein Themengebiet entgegen und erstellt anhand dessen einen eindeutigen Pfad, der auf ein vortrainiertes GPT2-Modell verweist.

- ‚TextGenerator': Ruft unter Zuhilfenahme des erzeugten Pfades ein

vortrainiertes GPT2-Modell auf und erzeugt eine auf Schwierigkeitsgrad sowie Themengebiet spezifizierte, textuelle Anforderungsbeschreibung inklusive Aufgabenstellung.

- ‚TextProcessor': Führt eine automatisierte Korrektur der Rechtschreibung durch.

Des Weiteren beinhaltet das Klassendiagramm das Paket ‚Models'. Darin befinden sich die Dateien für die GPT2-Modelle. Diese wurden in der deutschen Sprache vortrainiert und sind durch eigenes Trainingsmaterial auf den gewünschten Anwendungsfall zu spezialisieren. Da für die Nachfolger des Modells Lizenzkosten anfallen und die mit Studierenden durchgeführte Studie von Huber und Hagel (2022a) die Eignung von GPT2 in diesem Kontext aufzeigt, soll dieses auch weiterhin Anwendung finden. Für jeden Schwierigkeitsgrad und darunter jedes Themengebiet ist eine eigene Ordnerstruktur angelegt. Über die Klasse ‚ExerciseGenerator' werden diese Parameter an den ‚ModelPathGenerator' übergeben. Letzterer erzeugt einen Dateipfad, der den Aufruf eines entsprechenden Textgenerators ermöglicht. Das Training dieser erfolgte, wie im Abschnitt 4.3.1 und bei Huber und Hagel (2022a) beschrieben. Mithilfe der Fassadeklasse können Schwierigkeitsgrad und Themengebiet auch von Extern (beispielsweise einer Webseite) entgegengenommen und die erstellte Übungsaufgabe kommuniziert werden.

Für die Auswahl stehen die drei Schwierigkeitsgrade einfach, moderat und schwer zur Verfügung. Je nach Auswahl unterscheiden sich die Aufgabentexte wie folgt voneinander ($\hat{=}$ der Komplexitätsmetrik der Übungsaufgabe):

- **Einfach:** Die Anforderungsbeschreibung ist prägnant und umfasst lediglich wenige Sätze, in denen eine kleine Anzahl von Klassen mit Attributen, Methoden sowie Beziehungen beschrieben sind. Das resultierende Klassendiagramm ist überschaubar und wenig komplex. Es sind keinerlei Beschreibungen von Entwurfsmustern enthalten.

- **Moderat:** Die Anforderungsbeschreibung ist umfangreicher und beinhaltet Klassen, Attribute, Methoden sowie deren Beziehungen. Das resultierende Klassendiagramm beinhaltet einige Klassen mit unterschiedlichen Relationen zueinander. Es sind keinerlei Beschreibungen von Entwurfsmustern enthalten.

- **Schwer:** Die Anforderungsbeschreibung beinhaltet eine große Anzahl unterschiedlicher Klassen mit zugehörigen Attributen, Methoden und Beziehungen. Das resultierende Klassendiagramm ist weniger überschaubar und Studierende müssen sicherstellen, alle angeführten Konzepte zu berücksichtigen und diese (im Falle von Attributen sowie Methoden) korrekt zuzuordnen. Weiterhin sind Entwurfsmuster beschrieben, die es in das UML-Klassendiagramm zu integrieren und mit vorhandenen Klassen zu verknüpfen gilt.

Zusätzlich sind vier Themengebiete wählbar. Diese sind durch den Autor selektiert und stellen beispielhafte, informatiknahe Studiengänge dar. Bei Verfügbarkeit von Trainingsdaten ist diese Vorgehensweise um beliebige, inhaltliche Schwerpunkte erweiterbar. Je nach Variante unterscheiden sich die Aufgabentexte in ihrer Thematik. So wären für die Auswahlmöglichkeit „Informatik - Game Engineering“ beispielsweise Inhalte eines Computerspiels erläutert. Aufgrund der benötigten Menge an Trainingsdaten wurden im Rahmen dieser Arbeit nur Modelle für diesen Themenschwerpunkt trainiert. Da das Verfahren simultan auf beliebige Sachverhalte erweiterbar ist, besteht in diesem Zusammenhang kein zusätzlicher, wissenschaftlicher Beitrag durch die Bereitstellung weiterer Trainingsdaten für andere Themenschwerpunkte. Die resultierende Anforderungsbeschreibung ist vergleichbar mit der in Abbildung 4.2. Innerhalb der Aufgabentexte gibt es einige Signalworte, die sowohl für die Studierenden als auch für die automatisierte Erstellung einer Musterlösung hilfreich sein können. Tabelle 4.1 fasst diese zusammen. Sie sind durch den Autor auf Basis eines empirischen Standpunkts nach Ortner (2002) konstruiert. Demnach werden „[...] die Verhältnisse (Sachverhalte) in einem Anwendungsbereich lediglich so, wie sie angetroffen (oder von den Anwendern mitgeteilt) werden, in die Struktur (Architektur) einer Anwendungslösung umgeschrieben.“ (Ortner, 2002; S. 39). Zwar beschreibt Ortner (2002) weiter, dass Strukturwörter bezogen auf die formale Wahrheit respektive Korrektheit unpräzise gebraucht werden können, jedoch schließen die bereitzustellenden Fachtexte solche Unklarheiten aus. Seine Ansicht, dass Sprache im Kontext der Informatik eine „[...] Menge der in einer Grammatik erzeugbaren Ausdrücke (Sprachartefakte) [...]“ (Ortner, 2002; S. 45) ist, wird hier geteilt. Die zu verwendende Grammatik bewegt sich demnach in einem restriktiven Rahmen. Tabelle 4.1 beinhaltet Vokabeln, die

für eine Beschreibung notwendiger Konzepte (beispielsweise Vererbungsbeziehungen) relevant sind. Zu den Signalworten „Ist ein“ und „Sind“ muss angemerkt werden, dass diese nicht immer eindeutig auf eine Vererbungsbeziehung hinweisen. Das sogenannte *Liskov Substitution Principle* definiert, dass Instanzen einer Elternklasse durch die beliebiger Kindklassen ersetzbar sein müssen, ohne Änderungen am Programmcode erforderlich zu machen (vgl. Liskov, 1987; vgl. Liskov und Wing, 1994; vgl. Martin, 2018). Demnach wäre die Modellierung des Beispielsatzes: „Ein Quadrat ist ein Rechteck“ mit dem Rechteck als Elternklasse und den Quadrat als Kindklasse als falsch zu bewerten. Logischerweise enthält eine Klasse ‚Rechteck‘ je ein Attribut für die Seitenlänge als auch für die Seitenbreite. Erbt eine Klasse ‚Quadrat‘ von der Klasse ‚Rechteck‘, so erbt das Quadrat folgerichtig beide Attribute. Per Definition müssen Quadrate jedoch gleichseitig sein und sollten daher nur einen Parameter für die Festlegung aller Seitenlängen zur Verfügung stellen. Bei der Erzeugung der Trainingsdaten für die Textgeneratoren ist sicherzustellen, dass nur Klassen beschrieben sind, deren Vererbungsbeziehungen das Liskov Substitution Principle nicht verletzen. Weiterhin ist zu erwähnen, dass Klassennamen sowie Attribute in Anforderungsbeschreibungen konventionell durch Nomen beschrieben und Methoden anhand von Verben erkennbar sind (vgl. Seidl u. a., 2015). Durch diese Konzeption sind diese Gegebenheiten berücksichtigt.

Das in Abbildung 4.5 einsehbare Softwaredesign ist möglichst modular gehalten. Demnach sind einzelne Subtasks austauschbar beziehungsweise aufrufbar. Zusätzliche Funktionalitäten lassen sich demnach durch ein einfaches Austauschen oder Hinzufügen weiterer Klassen realisieren.

Wie von Huber und Hagel (2022a) beschrieben, erfolgt nach der Erstellung einer textuellen Anforderungsbeschreibung die Integration einer vorgefertigten Aufgabenstellung, die simultan zu den Vorgaben der Autoren formuliert ist (siehe Unterunterabschnitt 4.3.1.1). In Kombination ergeben sie eine vollständige Übungsaufgabe.

Die angeführte Konzeption erweitert den von Huber und Hagel (2022a) erläuterten Prototyp in einigen Punkten. So war dieser zuvor als eigenständige Anwendung nicht ohne Weiteres in ein Gesamtsystem integrierbar. Weiterhin gab es keinerlei Option für die Erstellung kontextbasierter Anforderungsbeschreibungen mit unterschiedlichen Schwierigkeitsgraden und

Signalwort(e)	**Erläuterung**
Attribut	Zeigt auf, dass ein oder mehrere auf das Signalwort folgende Nomen als Attribut(e) und nicht als Klassennamen identifiziert werden müssen.
Methode	Weist darauf hin, dass ein oder mehrere aufeinanderfolgende Verben als Methoden identifiziert werden müssen.
Entwurfsmuster	Gibt Aufschluss darüber, dass ein vorgelagertes oder darauffolgendes Nomen (beispielsweise „Dekorierer“) den Namen eines zu modellierenden Entwurfsmusters beschreibt.
Vom Typ	Deutet auf den Datentyp (bei Attributen) respektive den Typ eines Rückgabewerts (bei Methoden) hin.
Public / Protected / Package / Private	Beschreiben die gängigen Sichtbarkeiten, mit denen Attribute und Methoden einer Klasse versehen sein können.
Abstrakt	Deutet darauf hin, dass eine Klasse abstrakt zu modellieren ist.
Ist ein	Beschreibt eine Vererbungsbeziehung zweier oder mehrerer Klassen.
Sind	Beschreibt eine Vererbungsbeziehung zweier oder mehrerer Klassen.
Aggregation	Zeigt auf, dass eine Aggregationsbeziehung zwischen zwei Klassen besteht.
Komposition	Deutet auf eine Kompositionsbeziehung zwischen zwei Klassen hin.

Tabelle 4.1: Signalworte innerhalb der automatisiert generierten Anforderungsbeschreibung.

```
from ExerciseGenerator.Task.ModelPathGenerator
                        import ModelPathGenerator
from ExerciseGenerator.Task.TextGenerator
                        import TextGenerator
from ExerciseGenerator.Task.TextProcessor
                        import TextProcessor

class ExerciseGenerator:

 def generate_exercise(self, level_of_difficulty,
                                        exercise_category):

  self.load_text_generation_model(level_of_difficulty,
                                        exercise_category)
  exercise_text = self.generate_exercise_text()
  exercise_text = self.process_exercise_text(exercise_text)

  return exercise_text
```

Codebeispiel 4.1: Beispielhafte Implementierung der Klasse ‚ExerciseGenerator‘.

Beschreibungen von Entwurfsmustern.

4.3.3 Implementierung

Im Folgenden sind Codebeispiele für die zuvor beschriebenen Klassen abgebildet. Diese sind nicht vollständig und zeigen nur die bedeutendsten Inhalte. Die Namensgebung für Klassen, Attribute etc. erfolgt in englischer Sprache.

Der in Codebeispiel 4.1 abgebildete Programmcode zeigt exemplarisch, wie die Klasse ‚ExerciseGenerator‘ implementiert werden kann. In einem ersten Schritt erfolgt der Import der zuvor beschriebenen Tasks, welche der Erstellung und Korrektur der textuellen Anforderungsbeschreibung sowie der Aufgabenstellung dienen. Weiterhin beinhaltet die Klasse die Methode ‚generate_exercise(...)‘, der jeweils ein String für den gewünschten Schwierigkeitsgrad und das gewünschte Themengebiet übergeben wird. Darin erfolgt der Aufruf von drei klasseninternen Methoden, die wiederum eine Instanz der unterschiedlichen Tasks erzeugt und deren individuelle ‚run()‘-Methode ausführt. Das Endresultat stellt eine automatisiert generierte Übungsaufgabe dar. Letztere ist für alle Anforderungsbeschreibungen gleich und muss daher nicht von GPT2 erzeugt werden.

Der Programmcode für die Klasse ‚ModelPathGenerator‘ ist in Codebeispiel 4.2 einsehbar. Wie Abbildung 4.5 verdeutlicht, erbt sie von der ab-

```
import os
from ExerciseGenerator.Task.Task import Task

class ModelPathGenerator(Task):

def run(self):

 self.join_model_path()

 return self.__model_path

def join_model_path(self):

 self.__model_path = os.path.join(self.__model_path,
 self.__level_of_difficulty, self.__exercise_category)
```

Codebeispiel 4.2: Beispielhafte Implementierung der Klasse ‚ModelPathGenerator'.

```
from transformers import pipeline
from ExerciseGenerator.Task.Task import Task

class TextGenerator(Task):

 def run(self):

  generator = pipeline('text-generation',
                       max_length=self.__max_length,
                       model=self.__text_generation_model_path,
                       tokenizer=self.__tokenizer)

  return generator(self.__first_word)[0]['generated_text']
```

Codebeispiel 4.3: Beispielhafte Implementierung der Klasse ‚TextGenerator'.

strakten Klasse ‚Task' und beinhaltet folgerichtig eine ‚run()'-Methode. Darin enthalten ist ein zusätzlicher Aufruf der Methode ‚join_model_path()', welche einen individuellen Dateipfad zu einem Textgenerator in Abhängigkeit zu dem gewählten Schwierigkeitsgrad sowie dem gewählten Themengebiet erzeugt. Besagter Pfad wird in dem privaten Attribut ‚model_path' gespeichert, welches wiederum den Rückgabewert der ‚run()'-Methode darstellt.

Codebeispiel 4.3 beinhaltet einen Auszug der Klasse ‚TextGenerator'. In der darin implementierten ‚run()'-Methode findet das Einladen des Textgenerators mithilfe des zuvor generierten Dateipfades statt. Dieser Prozess greift auf die transformers-Bibliothek[21] zurück. Im ‚return'-Statement erfolgt ein Aufruf des Textgenerators, der anschließend eine textuelle An-

[21] Verfügbar unter: https://pypi.org/project/transformers/2.1.0/

```
from spellchecker import SpellChecker
from ExerciseGenerator.Task.Task import Task

class TextProcessor(Task):

 def run(self):

  spell_checker = SpellChecker(language="de",
                               case_sensitive=True)

  ...

  return exercise_text
```

Codebeispiel 4.4: Beispielhafte Implementierung der Klasse ‚TextProcessor'.

forderungsbeschreibung erzeugt, welche an die Klasse ‚ExerciseGenerator' zurückgegeben wird.

Als letztes ist die Klasse ‚TextProcessor' zu nennen. Der Programmcode ist in Beispiel 4.4 dargestellt. Unter Zuhilfenahme der Python-Bibliothek pyspellchecker[22] erfolgt eine automatisierte und wortweise Rechtschreibkorrektur innerhalb der ‚run()'-Methode. Deren Rückgabe stellt die korrigierte, textuelle Anforderungsbeschreibung dar.

4.3.4 Evaluation

Die in Unterabschnitt 4.3.1 enthaltene Vorarbeit von Huber und Hagel (2022a) untersucht mithilfe eines Fragebogens die Eignung nach diesem Verfahren automatisiert erzeugter Übungsaufgaben für die hochschulische Lehre von UML-Klassendiagrammen. Eine erneute Evaluation mit gleicher Fragestellung ist daher als nicht sinnvoll zu erachten, da die angeführte Konzeption und der resultierende Softwareprototyp dieses Verfahren aufgreifen sowie optimieren. Wie Abbildung 4.2 verdeutlicht, beinhalten die von Huber und Hagel (2022a) generierten, textuellen Anforderungsbeschreibungen teils grammatikalische Fehler sowie unverständliche Sätze, was manuelle Anpassungen erforderlich macht.

Diese Evaluation soll daher untersuchen, ob kontextbezogene Texte mit unterschiedlichen Schwierigkeitsgraden fehlerfrei generierbar sind. Dies impliziert eine Überprüfung der Erfüllung in der Konzeption berücksichtigter Anforderungen. Da die Aufgabenstellung unverändert bleibt, stehen die textuellen Anforderungsbeschreibungen im Fokus der Untersuchung.

[22] Verfügbar unter: https://pypi.org/project/transformers/2.1.0/

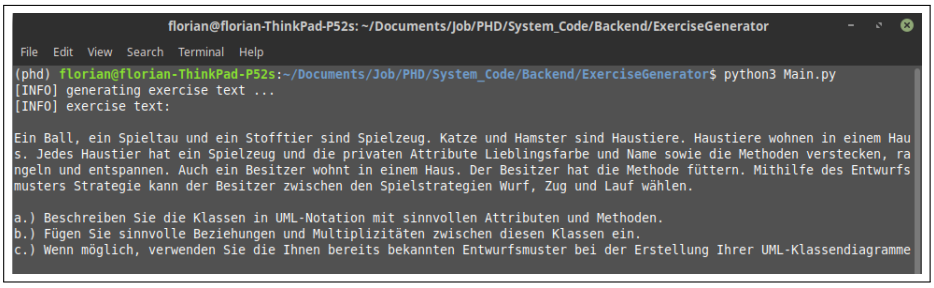


Abbildung 4.6: Im Kontext der Evaluation exemplarische und automatisiert erzeugte Übungsaufgabe.

Ein Aufruf der in dem Softwarepaket befindlichen Fassadeklasse (welche wiederum die beschriebenen Tasks aufruft) erfolgt durch eine hinzugefügte Klasse mit einer ‚main()'-Methode. Diese dient lediglich den Evaluationszwecken und wird anschließend wieder aus dem Paket entfernt. Insgesamt wurden zehn Aufgabentexte erzeugt. Drei mit Schwierigkeitsgrad ‚Einfach', drei mit Schwierigkeitsgrad ‚Moderat' und vier mit Schwierigkeitsgrad ‚Schwer'. Da Letztere länger sind und eine komplexere Satzstruktur aufweisen, ist hier am wahrscheinlichsten mit grammatikalischen respektive Satzbaufehlern zu rechnen. Die Prüfung erfolgte durch den Autor unter Zuhilfenahme einer durch den DUDEN online bereitgestellte Software für Rechtschreibprüfung[23].

Abbildung 4.6 zeigt exemplarisch eine der automatisch erzeugten Aufgaben. Neben den beschriebenen Klassen mit Attributen, Methoden und Beziehungen zueinander ist auch das Strategie-Entwurfsmuster enthalten. Die textuelle Anforderungsbeschreibung enthält weder Rechtschreib- noch Grammatikfehler. Weiterhin sind keine nicht nachvollziehbaren Sätze enthalten. Für die anderen neun Beispiele galt dies ebenfalls.

Vollständig auszuschließend sind solche Fehler innerhalb der Texte jedoch nicht. Sind sie grammatikalischer Natur (wie im Beispiel von Abbildung 4.6) können die Studierenden in den meisten Fällen dennoch korrekte UML-Klassendiagramme ableiten. Zusammenhangslose oder unverständliche Sätze stellen eine Herausforderung dar. Um diese zu umgehen, sind zusätzliche Trainingsdaten erstellt und die Aufgabentexte, wie in der Studie von Huber und Hagel (2022a) von Studierenden gewünscht, gekürzt worden.

Wie zuvor erwähnt, liegt das Erkenntnisinteresse dieses Abschnitts in

[23]Verfügbar unter: https://www.duden.de/rechtschreibpruefung-online

der Konzipierung und prototypischen Bereitstellung eines Softwareartefakts, das die erhobenen funktionalen Anforderungen hinreichend erfüllt. Im Sinne der teilweisen Beantwortung der angeführten Forschungsfrage ist das Ergebnis folglich ausreichend. In einem nächsten, nicht in dieser Arbeit enthaltenen Schritt wäre eine linguistische Untersuchung der Übungsaufgaben und im Speziellen der textuellen Anforderungsbeschreibungen denkbar. Für Untersuchungen zur Textqualität gibt es in der einschlägigen Fachliteratur eine Vielzahl unterschiedlicher Methoden (vgl. Schriver, 1989; vgl. Nussbaumer, 1991; vgl. Sieber, 1994; vgl. Göpferich-Görnert, 2018). So lässt sich beispielsweise die Textverständlichkeit umfassend untersuchen (vgl. Schriver, 1989; vgl. Göpferich-Görnert, 2018). Für weiterführende Einblicke sei an dieser Stele auf die zitierte Fachliteratur verwiesen.

4.4 Generieren von UML-Klassendiagrammen auf Basis von textuellen Anforderungsbeschreibungen

Wie Forschungsfrage 1.1 verdeutlicht, liegt das Erkenntnisinteresse dieses Kapitels nicht ausschließlich auf der automatisierten Erzeugung von textuellen Anforderungsbeschreibungen mit zugehörigen Aufgabenstellungen für Studierende. Es adressiert weiterhin die Problematik der Erzeugung von Musterlösungen zu vorliegenden Aufgabentexten. Dieser Abschnitt nimmt sich besagter Thematik an. In diesem Sinne ist eine Zusammenfassung einer geleisteten Vorarbeit sowie einer Konzipierung und prototypischen Implementierung eines Softwareartefakts vorgesehen. Abschließend findet sich eine Evaluation des Resultats. In Kombination der hier gewonnenen Erkenntnisse mit denen des vorherigen Abschnitts zur automatisierten Erzeugung von Übungsaufgaben lässt sich Forschungsfrage 1.1 beantworten. Die beschriebenen Inhalte sind folglich ein zentraler Bestandteil für die Erreichung der Zielsetzung der Arbeit.

4.4.1 Vorarbeit

Nach Bearbeitung der textuellen Aufgabe stellt eine Lehrperson üblicherweise eine Musterlösung bereit und bespricht diese ggf. mit den Studierenden (vgl. Huber und Hagel, 2022a; vgl. Huber und Hagel, 2022c). Sie dient als orientierungsgebendes Beispiel für einen möglichen Lösungsweg. Für Dozierende gestaltet sich nicht nur die Erstellung von Übungsaufgaben, sondern auch die Anfertigung einer zugehörigen Musterlösung zeitaufwendig (vgl. Huber und Hagel, 2022a; vgl. Huber und Hagel, 2022c). Da individuelles Feedback in diesen Lehrsituationen jedoch meist nicht möglich ist, fordern Studierende eine solche Musterlösung ein, da sie andernfalls nicht feststellen können, ob eine Aufgabe von ihnen richtig bearbeitet wurde.

Eine automatisierte Erstellung solcher Musterlösungen würde viel Zeit vonseiten der Dozierenden einsparen. Die Schwierigkeit hierbei ist jedoch, dass es sich um Texte in natürlicher Sprache handelt und die darin beschriebenen Diagrammkomponenten nicht ohne Weiteres extrahierbar sind. Eine (unter anderem) durch den Autor verfasste Publikation nimmt sich dieser Herausforderung an. Die darin enthaltene Vorgehensweise ist in Abbildung 4.7 dargestellt. Für die Implementierung setzten Huber und Hagel (2022c) auf die Python-Programmiersprache. Getestet wurde der Prototyp mit einem Aufgabentext in deutscher Sprache, der für Studierende im Rahmen der Software Engineering-Kurse der Hochschule für angewandte Wissenschaften Kempten (Deutschland) erstellt wurde. Dieser ist in Abbildung 4.8 einsehbar. Die darin enthaltenen Diagrammbestandteile sind mit den Farben rot (für Klassen), blau (für Attribute) und grün (für Methoden) markiert (vgl. Huber und Hagel, 2022c).

Die von Huber und Hagel (2022c) erörterten Schritte für eine automatisierte Erstellung von UML-Klassendiagrammen anhand unstrukturierter Texte sind im Folgenden aufgelistet und detailliert beschrieben (in Anlehnung an (vgl. Huber und Hagel, 2022c, S. 2). Übersetzung durch den Autor):

- **Schritt 1** Import des Aufgabentexts:
  Der Prozess beginnt mit dem Einlesen des Aufgabentexts. Nach diesem Schritt steht dieser als String zur Weiterverarbeitung bereit.

- **Schritt 2** Laden des Sprachmodells:
  Da die Aufgabentexte in diesem Lehrkontext in natürlicher Sprache

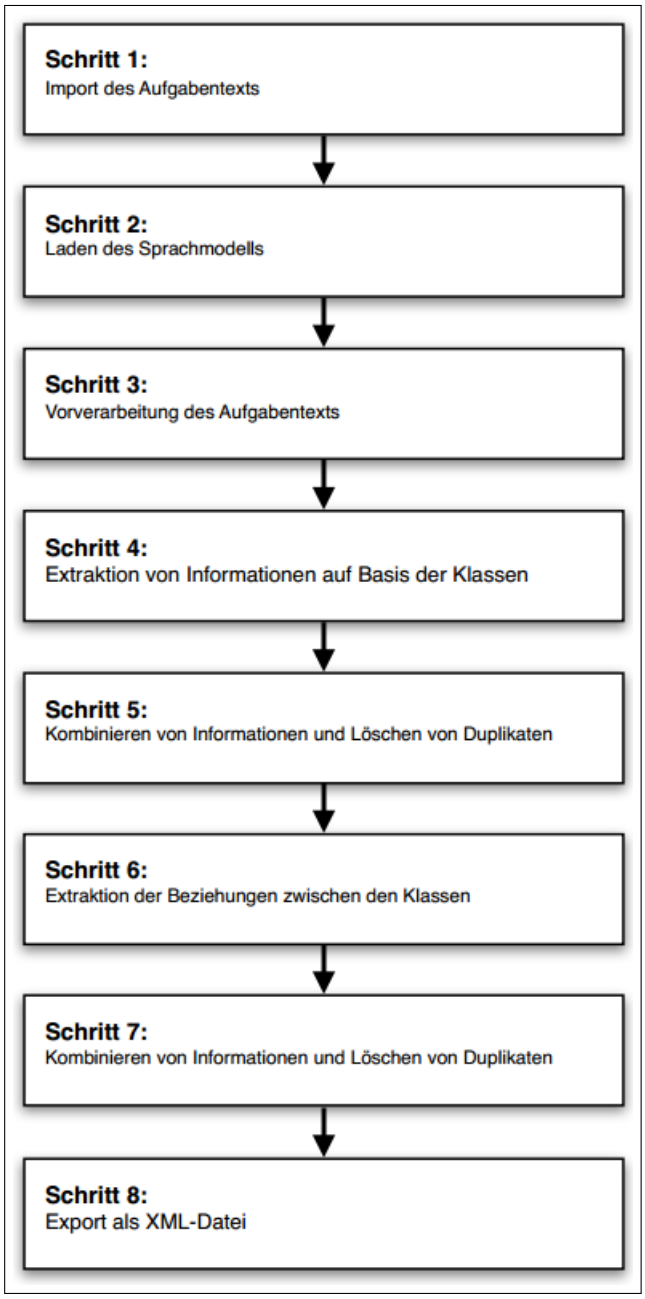


Abbildung 4.7: Vorgehen bei der automatisierten Generierung von UML-Klassendiagrammen anhand von Texten in natürlicher Sprache für die hochschulische Lehre von UML-Klassendiagrammen nach der Veröffentlichung von Huber und Hagel (2022c). Die Abbildung entstand in Anlehnung an (Huber und Hagel, 2022c, S. 2). Übersetzung durch den Autor.

Ein Spieler meldet sich mit Name und Passwort beim Spiel an. Dann kann er aus drei verschiedenen Charakteren seine Spielfigur wählen: Zauberer, Fee, Zwerg. Diese können zwei Waffen besitzen, die der Spieler aus der Menge Axt, Lanze, Dreizack und Pfeil und Bogen wählen kann. Waffen haben einen Schadenswert und einen Verteidigungswert. Außerdem kann ein Character eine Tasche mitführen, in der verschiedene Gegenstände gesammelt werden können, die man im Laufe des Spiels findet (Zauberbuch, Stein, Feder, Urkunde). Feinde des Character können Ork, Magier und Ritter sein. Diese können ebenfalls bis zu zwei der obigen Waffen besitzen.

Abbildung 4.8: Beispielhafte textuelle Anforderungsbeschreibung, die von Huber und Hagel (2022c) für die automatische Generierung von UML-Klassendiagrammen zugrundegelegt wurde. Die Abbildung wurde aus (Huber und Hagel, 2022c, S. 2) entnommen.

vorliegen, bietet sich die Verwendung eines NLP-Modells an. Huber und Hagel (2022c) setzen dabei auf die open-source Bibliothek spaCy[24]. Sie dient der Analyse komplexer Texte in natürlicher Sprache. Mithilfe der Bibliothek lassen sich Worttypen (beispielsweise Substantive oder Verben) feststellen, aber auch die Art der Beziehung zweier oder mehrerer Worte zueinander. Vortrainierte Modelle stehen für die Analyse von Texten in unterschiedlichen Sprachen bereit. Im Rahmen der Veröffentlichung wurde ein auf die deutsche Sprache trainiertes Modell herangezogen.

- **Schritt 3** Vorverarbeitung des Aufgabentexts:
  Die Vorverarbeitung von Texten stellt einen wichtigen Schritt in der Analyse natürlichsprachiger Sätze dar. Wie Abbildung 4.8 aufzeigt, können beispielsweise Klassennamen in unterschiedlichen Schreibweisen auftreten. Hier ist das Wort „Charakter“ einmal in deutscher und einmal in englischer Schreibweise („Character“) angeführt. Die Vorverarbeitung ermöglicht den Umgang mit diesen Sachverhalten und sorgt im genannten Beispiel dafür, dass nur eine Klasse für das Wort „Charakter“ angelegt wird. Weiterhin sind Pronomen in aufeinanderfolgenden Sätzen mit dem eigentlichen Klassennamen zu tauschen. Aus *Ein Zauberer hat einen Zauberstab. Außerdem kann er fliegen* wird *Ein Zauberer hat einen Zauberstab. Außerdem kann Zauberer fliegen.* Für das entwickelte Modell ist somit ersichtlich, auf welche Klasse sich ein bestimmter Satz bezieht.

[24]Einsehbar unter: https://spacy.io/

- **Schritt 4** Extraktion von Informationen auf Basis der Klassen:
  Abbildung 4.9 zeigt auf, wie Klasseninformationen (konkret: Klassennamen, Attribute und Methoden) mithilfe von spaCy auf Basis der einzelnen Sätze extrahierbar sind. So weisen Substantive beispielsweise auf Attribute oder Klassen hin. Anhand von Wortbeziehungen lassen sich diese voneinander differenzieren. Das Ergebnis stellt eine Liste mit Klassennamen und zugeordneten Attributen und / oder Methoden dar. Da alle Texte Satzweise analysiert werden, kann derselbe Klassenname in der Liste mehrfach vorhanden sein.

- **Schritt 5** Kombinieren von Informationen und Löschen von Duplikaten:
  Die in der Liste enthaltenen Informationen werden klassenweise miteinander kombiniert und Duplikate entfernt. So entsteht eine neue Liste, in der jeder Klassenname mit zugeordneten Attributen sowie Methoden lediglich einmal auftritt.

- **Schritt 6** Extraktion von Beziehungen zwischen den Klassen:
  Mithilfe der von spaCy detektierten Beziehungen zwischen den einzelnen Worten und anhand ausgewählter Phrasen lassen sich die Beziehungskomponenten des zu erstellenden Klassendiagramms regelbasiert extrahieren. Wie bei Schritt 4 erfolgt eine satzweise Analyse des Textes.

- **Schritt 7** Kombinieren von Informationen und Löschen von Duplikaten:
  Die Liste detektierter Beziehungskomponenten kann Duplikate enthalten, falls Beziehungen zwischen Klassen innerhalb des Aufgabentexts mehrfach beschrieben sind. In diesem Schritt werden diese Informationen miteinander kombiniert, um Duplikate zu entfernen.

- **Schritt 8** Export als XML-Datei:
  In einem letzten Schritt erfolgt der Export der extrahierten Informationen in eine XML-Datei. Diese kann durch gängige Modellierungseditoren geöffnet und das UML-Klassendiagramm eingesehen werden. Exemplarisch zeigten die Autoren dies mithilfe der Modellierungssoftware Enterprise Architect. Das Ergebnis ist in Abbildung 4.10 einseh-

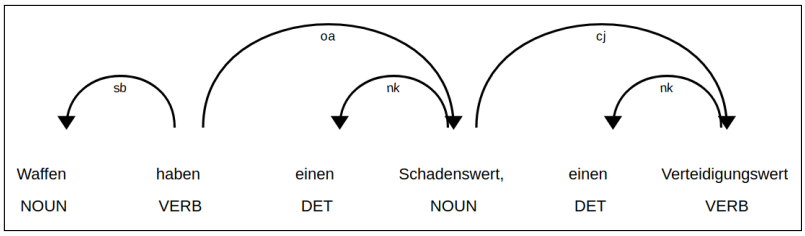


Abbildung 4.9: Beispielhafte Analyse eines Satzes im Kontext der Extraktion von Klasseninformationen. Die Abbildung wurde aus (Huber und Hagel, 2022c, S. 2) entnommen.

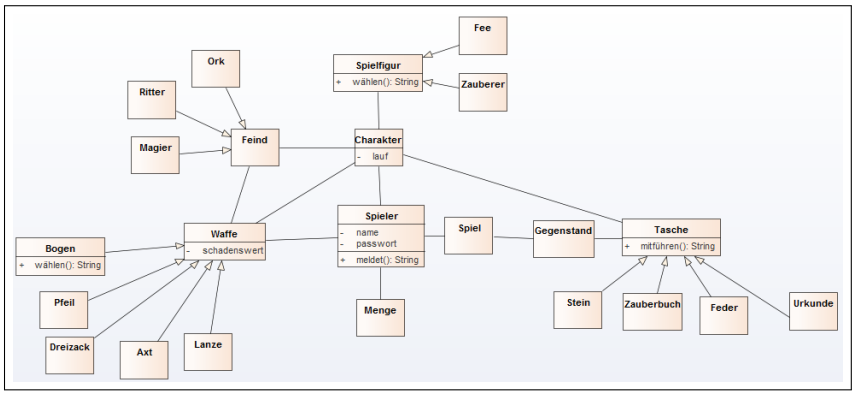


Abbildung 4.10: Beispielhaftes Ergebnis für die automatisierte Erstellung von UML-Klassendiagrammen anhand eines vorliegenden Aufgabentexts. Die Abbildung wurde aus (Huber und Hagel, 2022c, S. 3) entnommen.

bar.

Wie Abbildung 4.9 verdeutlicht, kann es vorkommen, dass spaCy einzelne Worttypen nicht korrekt erkennt. In diesem Beispiel ist das Substantiv „Verteidigungswert“ als Verb klassifiziert. Bei einer automatisierten Erstellung eines UML-Klassendiagramms resultiert daraus, dass dieses Substantiv nicht als Attribut zu einer Klasse zugeordnet werden kann.

Das Ergebnis des achtstufigen Verfahrens (siehe Abbildung 4.10) beinhaltet nicht alle im Aufgabentext beschriebenen Diagrammkomponenten und deren korrekten Inhalte. Auch Huber und Hagel (2022c) geben an, dass das entwickelte Modell zukünftig verbessert werden muss, um in der hochschulischen Lehre einsetzbar zu sein. Hier ist es von großer Relevanz, dass die automatisiert erstellten Musterlösungen korrekt sind.

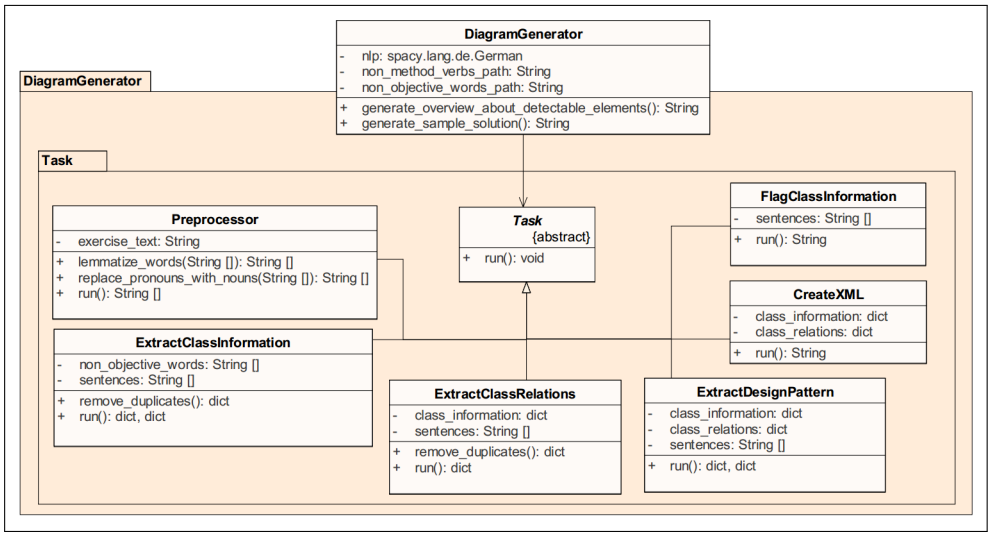


Abbildung 4.11: Softwaredesign der Softwarekomponente für die automatisierte Generierung von UML-Klassendiagrammen anhand textueller Anforderungsbeschreibungen.

4.4.2 Konzeption

Die von Huber und Hagel (2022c) publizierte Vorarbeit soll auch in der Konzeption dieser Softwarekomponente als Ausgangsbasis dienen. Im Sinne des DSR-Forschungsdesigns wird auch hier zyklisch verfahren (vgl. Hevner u. a., 2004). Da es sich bei der Publikation um ein Work-In-Progress handelt, ist implizit klar, dass der vorgestellte Prototyp noch nicht vollumfänglich in der hochschulischen Lehre integrierbar ist. Wie die Autoren beschreiben, wurden bisher noch nicht alle Diagrammbestandteile korrekt aus den textuellen Anforderungsbeschreibungen extrahiert (vgl. Huber und Hagel, 2022c). Weiterhin erwähnen die Autoren nicht, ob Sichtbarkeiten von Attributen respektive Methoden detektierbar sind. Auch die Extraktion von Entwurfsmustern ist in der Publikation nicht erwähnt. Ziel ist daher die Entwicklung von Regelwerken sowie die Modellierung einer Softwarekomponente zur Realisierung dieser.

Abbildung 4.11 visualisiert das konzipierte Softwaredesign. Für die jeweiligen Klassen sind die wichtigsten Attribute und Methoden abgebildet. Grundlegend folgt das Design dem Fassade-Entwurfsmuster (vgl. Eilebrecht und Starke, 2019; vgl. Kleuker, 2018). Das Paket ‚DiagramGenerator' beinhaltet alle dargestellten Diagrammbestandteile. Die eigentliche Programmlogik ist in dem Unterpaket ‚Task' zu finden. Die Kommunikationsschnitt-

stelle zu dieser stellt die Fassadeklasse ‚DiagramGenerator' dar, welche beispielsweise eine textuelle Anforderungsbeschreibung entgegennehmen und das Ergebnis der Erstellungsschritte zurückliefern kann. Entsprechend den Prinzipien der Trennung von Zuständigkeiten (vgl. Vogel u. a., 2011) und der eindeutigen Verantwortung (vgl. Martin, 2018) erhält jede Klasse eine individuelle Aufgabe. Aufgrund dieses modularen Aufbaus können bestehende Diagrammkomponenten flexibel angepasst oder neue hinzugefügt werden. Die abstrakte Klasse ‚Task' beinhaltet eine ‚run()'-Methode und vererbt diese an alle Kindklassen. Sie stellen den folgenden Funktionsumfang bereit:

- ‚Preprocessor': Unterteilt eine textuelle Anforderungsbeschreibung in ihre individuellen Sätze und speichert diese als Liste. Beginnt ein Listenelement mit einem Personalpronomen, wird versucht dieses, wie von Huber und Hagel (2022c) erläutert, durch das eigentliche Nomen zu ersetzen. Da die Namensgebung innerhalb von UML-Klassendiagrammen konventionell im Singular erfolgt (vgl. Seidl u. a., 2015), führt die Klasse weiterhin eine Lemmatisierung aller Nomen durch. Dabei handelt es sich um eine automatisierte Konvertierung in die Grundform des jeweiligen Substantivs. Nach den Vorverarbeitungsschritten liegt die Anforderungsbeschreibung in einer auswertbaren Form vor.

- ‚ExtractClassInformation': Extrahiert alle definierten Diagrammbestandteile auf Klassenbasis (Klassennamen, Attribute, Methoden) sowie deren Sichtbarkeiten (public, protected, package und private) aus der vorverarbeiteten Anforderungsbeschreibung und speichert diese in einer Dictionary-Struktur[25].

- ‚ExtractClassRelations': Extrahiert alle beschriebenen Beziehungen zwischen den detektierten Klassen aus der vorverarbeiteten Anforderungsbeschreibung und speichert diese in einer separaten Dictionary-Struktur.

- ‚ExtractDesignPattern': Sofern ein Entwurfsmuster in der Anforderungsbeschreibung enthalten ist, erfolgt eine automatisierte Speiche-

[25]Eine Erklärung und ein Beispiel findet sich unter: https://docs.python.org/3/tutorial/datastructures.html

rung benötigter Klassen und Beziehungen in den zuvor angelegten Dictionary-Strukturen.

- ‚CreateXML‘: Generiert eine XML-Datei anhand aller extrahierten Informationen. Diese ist, wie von Huber und Hagel (2022c) erläutert, beispielsweise in der Modellierungssoftware Enterprise Architect importierbar und lässt sich dort sowohl bearbeiten als auch anzeigen.

- ‚FlagClassInformation‘: Erkennt und klassifiziert definierte Diagrammbestandteile in der vorverarbeiteten Anforderungsbeschreibung und markiert diese mithilfe von HyperText Markup Language (HTML) Annotationen. Dies ermöglicht es Klassen, Attribute, Methoden und Namen von Entwurfsmustern auf Weboberflächen visuell hervorzuheben. In Abschnitt 4.5 wird dies mithilfe eines Beispiels verdeutlicht. Die Klasse dient der reinen Markierung beschriebener Konzepte. Es erfolgt keine Speicherung von Informationen und keinerlei Konvertierung in eine XML-Datei.

Die von Huber und Hagel (2022c) präsentierte und in Unterabschnitt 4.4.1 beschriebene Vorgehensweise sowie Technologieauswahl soll auch hier Anwendung finden. Die Python-Bibliothek spaCy ist für komplexe Analysen und Auswertungen von Texten gut geeignet. In der Literatur waren durch den Autor keine Vorgaben für Extraktion von UML-Klassendiagrammkomponenten aus textuellen Anforderungsbeschreibungen in deutscher Sprache auffindbar. Daher ist die Entwicklung eigener Regelwerke notwendig. Diese sind in den Unterunterabschnitten 4.4.2.1, 4.4.2.2 sowie 4.4.2.3 detailliert angeführt und geben Aufschluss über den Extraktions- sowie Umwandlungsprozess. Die Informationsextraktion basiert auf der Analyse einzelner Sätze. Da Klassen über mehrere Sätze beschrieben sein können, erfolgt nach Ausführung der Regelwerke jeweils eine Zusammenführung. Implementiert sind diese in den Klassen ‚ExtractClassInformation‘, ‚ExtractClassRelations‘ und ‚ExtractDesignPattern‘. Es ist anzumerken, dass die darin codierten Regeln keine Allgemeingültigkeit aufweisen können. Die kontextuell relevante Fachtextsorte besitzt in jeder Hinsicht klare sprachliche Vorgaben und schränkt die Komplexität der Untersuchung daher bewusst ein, wodurch eine Verarbeitung der Inhalte überhaupt erst möglich wird. Das Erkenntnisinteresse orientiert sich speziell an diesen Maßgaben. Ziel der Regelwerke

kann demnach nicht die Informationsextraktion frei formulierter Textpassagen mit keinerlei Einschränkungen sein. Vielmehr soll der Fokus auf der Entwicklung von Regelwerken liegen, welche eine zuverlässige Erstellung von UML-Klassendiagrammen anhand der zuvor beschriebenen textuellen Anforderungsbeschreibungen erlaubt. Letztere müssen somit definierten Richtlinien (wie bspw. der Verwendung von Signalworten) folgen und dürfen außerdem keine Diagrammkomponenten enthalten, die in den resultierenden UML-Klassendiagrammen nicht enthalten sind. Weiterhin sind Relationen zwischen individuellen Klassen durch dafür vorgesehene Beziehungskomponenten und nicht als Attribute zu modellieren. Durch eine Abstimmung der Regelwerke auf die Anforderungsbeschreibungen wird eine Informationsextraktion möglich. Für die Lösung der Problematik wären andernfalls Textverständnis und Domänenwissen vonnöten. So kann beispielsweise das Substantiv ‚Ort' bei der Beschreibung einer Klasse ‚Adresse' als Attribut fungieren, in anderen Kontexten jedoch eine individuelle Klasse darstellen. Schon dieses einfache Exempel verdeutlicht, dass bei der (automatisierten) Generierung von UML-Klassendiagrammen keine allgemeingültigen und kontextfreien Regeln definierbar sind. Mithilfe der für die Textgeneratoren entwickelten Trainingsdaten ist sicherzustellen, dass keine beliebigen Freitexte erstellbar sind, sondern die Ergebnisse vielmehr einer definierten und auswertbaren Struktur in natürlicher Sprache entsprechen. In deutschsprachigen Übungstexten zu UML-Klassendiagrammen gibt es außerdem diverse sprachorientierte Modellierungstraditionen. So weisen beispielsweise Substantive meist auf Klassen- oder Attributnamen hin. Diese sind in den Regelwerken entsprechend berücksichtigt und lassen sich in der einschlägigen Fachliteratur (bspw. bei Seidl u. a. (2015)) nachlesen.

Sind alle in der textuellen Anforderungsbeschreibung enthaltenen Diagrammkomponenten sowie deren Beziehungen zueinander identifiziert, lassen sich diese Informationen mithilfe eines im Standard bereitgestellten Moduls[26] der Python-Programmiersprache in eine XML-Datei überführen. Baumstrukturen können hierbei beliebig aufgebaut und angepasst werden. Die resultierende Datei ist in die Modellierungssoftware Enterprise Architect importierbar. Das Konzept ist jedoch auf beliebige Modellierungsanwendungen erweiterbar. Die XML-Datei kann als eine Musterlösung für gegebene

[26]Dokumentation einsehbar unter: https://docs.python.org/3/library/xml.html

Übungsaufgaben dienen.

Mithilfe der Klasse ‚FlagClassInformation' lassen sich Klassennamen, Attribute, Methoden und Namen von Entwurfsmustern innerhalb der textuellen Anforderungsbeschreibungen identifizieren. Sie gibt selbige in Form eines HTML-Strings zurück, der beschriebene Diagrammkomponenten mit Annotationen versehen hat. Der Import in eine Weboberfläche führt somit zu einer farblichen Hervorhebung dieser. Studierende können diese Funktionalität als orientierungsgebende Hilfestellung verwenden. Der Aufruf aller zuvor erläuterten und in Abbildung 4.11 dargestellten Klassen ist hierfür nicht notwendig. Über die Fassadeklasse ist daher eine separate Methode zur Instanziierung und Ausführung von ‚FlagClassInformation' bereitzustellen. Innerhalb des UML-Klassendiagramms ist diese in der Klasse ‚DiagramGenerator' als ‚generate_overview_about_detectable_elements()' modelliert.

4.4.2.1 Entwicklung eines Regelwerks für die Extraktion von Informationen auf Klassenbasis

ID	Beschreibung	Beispielsatz
HK1	Substantive weisen auf Klassennamen oder Attribute hin.	Ein mutiger **Spieler** meldet sich mit **Name** und **Passwort** beim **Spiel** an. (Siehe Abbildung 4.12)
HK2	Verben deuten auf Methodennamen hin.	Ein mutiger Spieler **meldet** sich mit Name und Passwort beim Spiel an. (Siehe Abbildung 4.12)
HK3	Besitzt ein Verb einen separierbaren Präfix, so muss eine Zusammenführung von Verb und Präfix sowie eine anschließende Lemmatisierung erfolgen.	Ein mutiger Spieler **meldet** sich mit Name und Passwort beim Spiel **an**. (Siehe Abbildung 4.12)
HK4	Handelt es sich bei einem Substantiv um ein nichtgegenständliches Nomen, welches in einer definierbaren Liste hinterlegt ist, wird dieses als Attribut klassifiziert.	Ein mutiger Spieler meldet sich mit **Name** und **Passwort** beim Spiel an. (Siehe Abbildung 4.12)
HK5	Gegenständliche Nomen und solche, die nicht als nichtgegenständliche Substantive identifizierbar sind, gelten als Klassenname, sofern es sich nicht um ein vordefiniertes Signalwort (siehe Tabelle 4.1) handelt.	Der **Spieler** hat eine Augenfarbe und eine **Waffe**.
HK6	Das Signalwort „Attribut“ weist darauf hin, dass ein oder mehrere folgende Substantive einer Klasse als Attribute zugewiesen werden müssen.	Der Spieler hat das **Attribut Augenfarbe**.

HK7	Das Signalwort „Methode“ weist darauf hin, dass ein oder mehrere folgende Verben einer Klasse als Methoden zugewiesen werden müssen.	Der Spieler hat die **Methode hüpfen**.
HK8	Steht ein Signalwort für die Sichtbarkeit (public, protected, package, private) vor einem als Attribut oder Methode klassifizierten Wort, wird diese entsprechend zugeordnet sowie gespeichert und ist folglich im resultierenden UML-Klassendiagramm enthalten.	Der Spieler hat das **private** Attribut **Augenfarbe** und die **protected** Methode **hüpfen**.
HK9	Steht ein Signalwort für eine Sichtbarkeit (public, protected, package, private) vor dem Signalwort „Attribut“ oder „Methode“, sind alle dadurch beschriebenen Attribute und Methoden mit dieser zu versehen.	Der Spieler hat die **private** Attribut **Augenfarbe** und **Haarfarbe**.
HK10	Die Signalworte „Vom Typ“ geben Aufschluss über den Typ eines Attributs oder den Rückgabetyp einer Methode.	Der Spieler hat einen **Gesundheitszustand vom Typ Integer**.
HK11	Das Signalwort „Abstrakt“ vor einem Klassennamen weist darauf hin, dass es sich um eine abstrakte Klasse handelt.	Die **Abstrakte** Klasse **Spieler** hat eine Waffe.

HK12	Falls sich ein als Methode identifiziertes Verb auf ein als Klassenname erkanntes Subjekt bezieht (es besteht eine Subjektbeziehung ausgehend vom Verb zum Substantiv), ist dieses der Klasse als Methode zuzuordnen.	Ein mutiger **Spieler meldet** sich mit Name und Passwort beim Spiel an. (Siehe Abbildung 4.12)
HK13	Falls ein Hilfsverb eine Subjektbeziehung zu einem Substantiv (welches als Klassenname identifiziert wurde) besitzt und eine Teilsatzbeziehung („clausal object“) zu einem Verb (welches als Methode identifiziert wurde) hat, so ist das Verb der Klasse als Methode zuordenbar. Regel HK3 kann zusätzlich in Kraft treten.	Dann **kann Spieler** aus drei verschiedenen Charakter **wählen**. (Siehe Abbildung 4.13)
HK14	Hilfsverben und in einer vordefinierten Liste enthaltene Verben sind nicht als Methodennamen zu identifizieren.	Eine Waffe **hat** einen Schadenswert und einen Verteidigungswert. (Siehe Abbildung 4.14)
HK15	Steht ein als Attribut deklariertes Subjekt als Akkusativobjekt zu einem Verb oder einem Hilfsverb und besteht von Letzteren eine Subjektbeziehung zu einem als Klasse identifiziertem Substantiv, so ist das Attribut der Klasse zuordenbar.	Eine **Waffe hat** einen **Schadenswert** und einen Verteidigungswert. (Siehe Abbildung 4.14)

HK16	Falls nach der Regel HK15 keine Zuordnung eines Attributs zu einer Klasse erfolgt ist, wird versucht, ein zugehöriges Verb, Hilfsverb oder den zugehörigen Präfix eines Verbs für dieses Substantiv zu erkennen und dessen Subjektbeziehung zu einem Klassennamen zu identifizieren.	Einen **Energiewert besitzen** die **Feind**.
HK17	Erfolgte eine Zuweisung eines Attributs zu einer Klasse, wird geprüft, ob nach dem Attributnamen weitere folgen (durch Kommata sowie das Wort „und“ identifizierbar) und ebenfalls dieser Klasse zugeordnet werden können. Folgt keiner dieser Satzbausteine, endet die automatisierte Zuordnung (ausgenommen hiervon sind Artikel, die keinen Abbruch auslösen).	Eine Waffe hat einen Schadenswert **und einen Verteidigungswert**. (Siehe Abbildung 4.14)
HK18	Erfolgte eine Zuweisung einer Methode zu einer Klasse, wird geprüft, ob nach dem Methodennamen weitere folgen (durch Kommata sowie das Wort „und“ identifizierbar) und ebenfalls dieser Klasse zugeordnet werden können. Folgt keiner dieser Satzbausteine, endet die automatisierte Zuordnung (ausgenommen hiervon sind Artikel, die keinen Abbruch auslösen).	Der Spieler hat die Methoden springen**, hüpfen und werfen**.

HK19	Ist keinerlei Zuordnung eines Attribut- oder Methodennamens möglich, erfolgt eine Suche nach bereits identifizierten Klassennamen, die in ihrer Position des Satzes links stehen. Ist auch dies nicht möglich, erfolgt eine Suche rechts der Satzposition des Attribut- oder Methodennamens.	Der **Spieler** kennt neben einem Name außerdem noch ein Passwort sowie eine definierte **Augenfarbe**.
HK20	Steht ein Adjektiv vor einem Substantiv, welches nicht als Attribut identifizierbar ist, so setzt sich der Klassenname aus beiden Wörtern zusammen.	Ein **elektrisches Auto** ist ein Fahrzeug. (Siehe Abbildung 4.15)

Tabelle 4.2: Regelwerk zur Extraktion von Informationen auf Klassenbasis aus textuellen Anforderungsbeschreibungen.

Wie in der Literatur (vgl. Seidl u. a., 2015) und der Konzeption für die Erstellung textueller Anforderungsbeschreibungen (siehe Unterabschnitt 4.4.2) erläutert, sind Klassennamen und Attribute konventionell als Substantive und Methoden als Verben erkennbar. Anhand dieser Gegebenheiten lassen sie sich in Aufgabentexten identifizieren. Wie Abbildung 4.12 zeigt, sind Worttypen (Substantiv, Verb etc.) sowie die Beziehungen von Worten innerhalb eines Satzes durch die spaCy-Bibliothek detektierbar. Bei den Substantiven „Spieler“, „Name“, „Passwort“ und „Spiel“ handelt es sich folglich um Klassennamen oder Attribute. Das Verb „meldet“ impliziert das Vorhandensein einer Methode. Tabelle 4.2 fasst alle Regeln für die Extraktion von Informationen auf Klassenbasis zusammen und beschreibt diesen Sachverhalt in HK1 und HK2. Zusätzlich enthalten sind Beispielsätze, anhand derer sich die Regeln nachvollziehen lassen. Die Substantive wurden einer Lemmatisierung unterzogen.

In der deutschen Sprache kann ein Verb einen separierbaren Präfix besitzen. In der Abbildung ist dies durch eine ‚svp‘-Beziehung (die Bibliothek verwendet englische Bezeichnungen und somit steht „svp“ in diesem Fall für „separable verb prefix“) erkennbar. Der eigentliche Methodenname

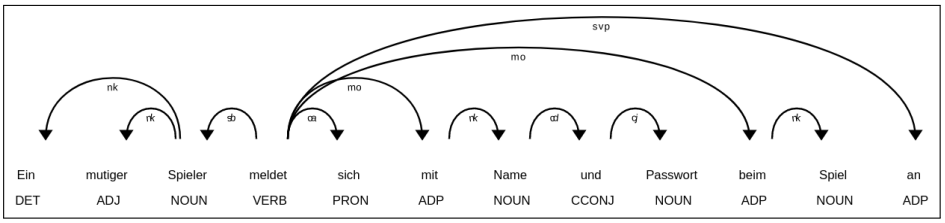


Abbildung 4.12: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem ersten Beispielsatz.

muss also „anmelden“ lauten, was die Detektion des Präfixes und eine anschließende Überführung des Wortes in seine Grundform (Lemmatisierung) erfordert. HK3 fasst dies in einer Regel zusammen. Da Substantive sowohl Klassennamen als auch Attribute repräsentieren können, ist eine Unterscheidung weder anhand des Worttyps noch der Wortbeziehungen möglich. Bei Attributen handelt es sich in der Regel um nichtgegenständliche Nomen. Als Beispiel hierfür sind „Name“ und „Passwort“ zu nennen. Ohne explizites Kontextwissen lassen sich diese nicht von Klassennamen abgrenzen. Die Konzeption inkludiert daher eine durch den Autor vordefinierte Liste nichtgegenständlicher Nomen, die immer als Attribute identifiziert und somit Klassen zugeordnet werden müssen. Eine Differenzierung der Typen von Nomen durch den Prototyp ist nicht vorgesehen und erfordert daher die Verfügbarkeit und das Pflegen einer solchen Auflistung. Auf Codeebene ist diese beliebig erweiterbar.

Gegenständliche Nomen wie beispielsweise „Spieler“ und „Spiel“ hingegen sind als Klassennamen zu speichern. Hiervon ausgenommen sind die in Tabelle 4.1 angeführten Signalwörter, die für die Modellierung zwar relevant sind, selbst aber nicht namentlich im Klassendiagramm vorkommen. HK4 und HK5 fassen dies zusammen.

Besagte Signalworte finden in der automatisierten Generierung eines Diagramms ebenfalls Berücksichtigung. Stehen nach dem Wort „Attribut“ ein oder mehrere Substantive, sind diese ungeachtet ihrer Gegenständlichkeit als Attribute zu klassifizieren (siehe HK6). Folgerichtig weisen Verben, die nach dem Wort „Methode“ stehen (aufgrund der Lemmatisierung in der Vorverarbeitung kann das Substantiv nicht im Plural auftreten) auf Methoden hin (siehe HK7). Stehen die Signalworte für Sichtbarkeiten (public, protected, package, private) vor einem Attribut- oder Methodennamen,

müssen diese entsprechend gespeichert und zugeordnet werden. Diese Regel findet sich in HK8. Steht eine Sichtbarkeit vor „Attribut“ oder „Methode“, wird diese allen dadurch beschriebenen Attributen und Methoden zugewiesen, wie in HK9 nachzulesen ist. Weiterhin zu nennen sind die Signalworte „Vom Typ“, welche Aufschluss über den Datentyp eines Attributs oder den Rückgabetyp einer Methode geben (siehe HK10). Steht „Abstrakt“ vor einem als Klassenname erkannten Substantiv, ist diese als abstrakte Klasse zu modellieren, wie in HK11 beschrieben ist.

Eine Zuordnung von Attributen und Methoden zu den jeweiligen Klassen ist anhand der Wortbeziehungen möglich. Das Beispiel in Abbildung 4.12 beinhaltet die Klassen „Spieler“ und „Spiel“. Die Methode „anmelden“ scheint daher zu einer der beiden Klassen zu gehören. Wie von spaCy erkannt, besteht eine Beziehung zwischen „meldet“ und „Spieler“. Konkret zeigt die Abbildung, dass für dieses Verb ein bestimmtes Nomen als Subjekt (durch spaCy mit „sb“ bezeichnet) ausgewiesen ist. Dadurch ist eine Zuordnung des Methodennamens möglich. Diese Regel findet sich in HK12. Ein weiteres Beispiel in Abbildung 4.13 zeigt, dass auch Hilfsverben, wie das Wort „kann“, eine Subjektbeziehung zu einem Nomen aufweisen kann. Zu dem Verb „wählen“ gibt es ebenfalls eine Beziehung (durch „oc“ für „clausal object“ in der Abbildung gekennzeichnet). Sie zeigt auf, welches Verb zu einem vorangegangenen Substantiv zugeordnet werden muss, damit der (Teil-)Satz inhaltlich sinnvoll ist. Somit ist nachvollziehbar, dass die Klasse „Spieler“ etwas „wählen“ kann und folglich mit dieser Methode versehen werden muss. Zusammengefasst ist dies in HK13 nachzulesen. Weiterhin sind Hilfsverben und in einer vordefinierten Liste enthaltene Verben nicht als Methodennamen zu identifizieren (siehe HK14). So würde das in einem weiteren Beispiel in Abbildung 4.14 angeführte Verb „hat“ der Klasse „Waffe“ nicht als Methode zugewiesen werden. Eine automatisierte Übergabe von Klassen als Methodenparameter ist nicht vorgesehen. Das Konzept ist dahingehend jedoch erweiterbar.

Abbildung 4.14 zeigt, dass Attributnamen auch als Akkusativobjekt (anhand der Abkürzung „oa“ erkennbar) zu einem Verb oder Hilfsverb stehen können. Besteht von letzteren eine Subjektbeziehung zu einem als Klasse festgelegtes Nomen, ist das Attribut dieser zuordenbar (siehe HK15). Ist nach wie vor keine Klasse für ein Attribut identifiziert, wird versucht, ein

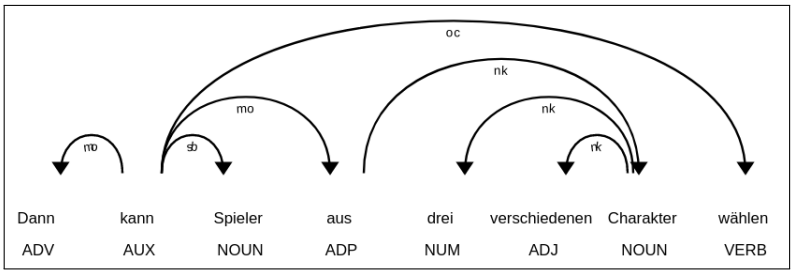


Abbildung 4.13: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem zweiten Beispielsatz.

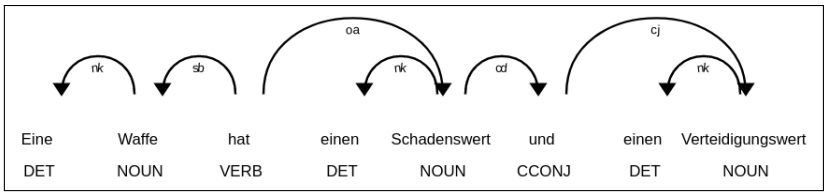


Abbildung 4.14: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem dritten Beispielsatz.

Verb, Hilfsverb oder der Präfix eines Verbs mit einer Verbindung zu diesem Substantiv zu detektieren und dessen Subjektbeziehung zu einem Nomen zu identifizieren (siehe HK16). Tritt eine der beschriebenen Regeln in Kraft, erfolgt eine automatisierte Prüfung, die bestimmt, ob eine Aufzählung weiterer Attributnamen folgt. Ist dies der Fall, können diese derselben Klasse zugewiesen werden. Die Substantive müssen dafür durch Kommata oder das Wort „und“ miteinander verknüpft sein. Folgt keiner der genannten Satzbausteine oder ein Artikel (im Beispiel von Abbildung 4.14 das Wort „einen“) auf einen Attributnamen, erfolgt für den Rest des Satzes eine erneute Zuordnung zu ggf. weiteren Klassennamen. HK17 fasst dies zusammen. Im Beispiel von Abbildung 4.14 würde durch besagten Regelsatz eine Zuordnung der Attribute „Schadenswert“ und „Verteidigungswert“ zu der Klasse „Waffe“ erfolgen. Für die Zuordnung von Methoden liegt die gleiche Vorgehensweise zugrunde (siehe HK18). Ist keinerlei Zuordnung eines Attributs oder Methodennamens möglich, erfolgt eine Suche nach Substantiven, die in der Position des Satzes links stehen. Ist auch dies nicht möglich, muss rechts der Position des Attributsnamens gesucht werden. Regel HK19 beschreibt diese Vorgehensweise.

Zusätzlich zu nennen sind beschreibende Adjektive, die vor Klassennamen auftreten. Diese können für die Benennung der Klasse relevant sein.

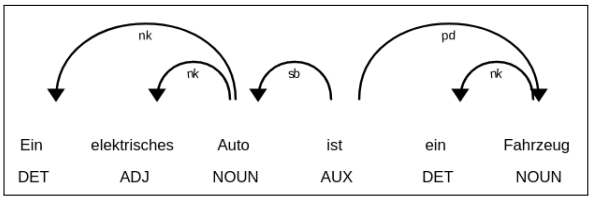


Abbildung 4.15: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem vierten Beispielsatz.

Das Beispiel in Abbildung 4.15 verdeutlicht dies. Die dort angeführte Kindklasse lediglich als „Auto“ zu titulieren, wäre nicht korrekt. Das Adjektiv „elektrische“ ist als Namenszusatz zu berücksichtigen, um den Klassennamen „ElektrischesAuto“ zu erhalten. Dies ist in Regel HK20 festgehalten. Die Annahme setzt voraus, dass es sich bei einem mit Adjektiv beschriebenen Substantiv um ein fundamental anderes Objekt handelt, für das eine separate Klasse benötigt wird. Die erstellten Aufgabentexte berücksichtigen dies entsprechend. Aufgrund der Einschränkung der Textsorte ist in solchen Fällen keine beliebige Komplexität der Sachverhalte zu erwarten. Die Lehrsituation schließt dies explizit aus, um Studierende nicht zu überfordern. Eine generelle Zuordnung beschreibender Adjektive zu Klassennamen ist folglich möglich.

Die Speicherung aller Klasseninformationen erfolgt in einer Dictionary-Struktur.

4.4.2.2 Entwicklung eines Regelwerks für die Extraktion von Beziehungen zwischen detektierten Klassen

Die Extraktion von Beziehungen zwischen Klassen wird in einem separaten Schritt durchgeführt, welcher der Detektion von Klasseninformationen bewusst nachgelagert ist. Bei Ausführung dieses Schrittes sind alle Klassen sowie deren Attribute und Methoden bekannt. Da solche Informationen über mehrere Sätze verteilt sein können, ist eine eindeutige Zuweisung von Beziehungen möglich, ohne diese in einem zusätzlichen Verarbeitungsvorgang zusammenführen zu müssen. Weiterhin sind nun lediglich Klassennamen relevant, was eine Berücksichtigung von Attributen und Methoden überflüssig macht. Für Sätze, die Beschreibungen von Entwurfsmuster enthalten, erfolgt eine gesonderte Betrachtung (siehe Unterunterabschnitt 4.4.2.3). Alle für das Regelwerk identifizierten Regeln sind in Tabelle 4.3 enthalten.

ID	Beschreibung	Beispielsatz
HB1	Ist lediglich eine einzige Klasse in einem Satz beschrieben, ist keine Beziehung erkennbar.	Ein **Spieler** hat einen Name. (Hinweis: „Name“ ist hierbei als Attribut identifiziert und stellt folglich keinen Klassennamen dar.)
HB2	Die Signalworte „Ist ein“ weist auf eine Vererbungsbeziehung hin. Der Klassenname, der in seiner Satzposition vor den Signalworten steht, stellt die Kindklassen dar (die nachgelagerte folgerichtig die Elternklasse).	Eine **Wasserfrau ist ein Wasserwesen**.
HB3	Das Signalwort „sind“ zeigt auf eine Vererbungsbeziehung hin. Klassennamen, die in ihrer Satzposition vor dem Signalwort stehen, stellen die Kindklassen dar (die nachgelagerte folgerichtig die Elternklasse). Grammatikalisch ist die Verwendung des Signalwortes sinnvoll, wenn multiple Kindklassen beschrieben sind.	**Wasserfrau** und **Wassermann sind Wasserwesen**.

HB4	Das Signalwort „Aggregation“ verdeutlicht, dass eine Aggregationsbeziehung von einer oder mehreren Klassen zu einer bestimmten Klasse besteht. Klassennamen, die in ihrer Satzposition vor dem Signalwort stehen, stellen bei der Teil-Ganzes-Beziehung die Teilkomponente dar (der nachgelagerte Klassenname folgerichtig das Ganze).	Der **Motor** hat eine **Aggregation** zum **Auto**.
HB5	Das Signalwort „Komposition“ zeigt auf, dass eine Kompositionsbeziehung von einer oder mehreren Klassen zu einer bestimmten Klasse besteht. Klassennamen, die in ihrer Satzposition vor dem Signalwort stehen, stellen bei der Teil-Ganzes-Beziehung die Teilkomponente dar (der nachgelagerte Klassenname folgerichtig das Ganze).	Der **Motor** hat eine **Komposition** zum **Auto**.
HB6	Stehen mehrere Klassennamen in einer Aufzählung hintereinander und wird eine Beziehung zu einer weiteren Klasse erkannt, so gilt diese Relation für alle Klassen in der Aufzählung.	**Wasserfrau, Wassermann und Wassergeist sind Wasserwesen**.

HB7	Besitzt ein Hilfsverb eine Subjektbeziehung zu einem Klassennamen und weiterhin eine Akkusativobjekt-Beziehung zu einem weiteren Klassennamen, besteht eine Assoziation zwischen diesen Klassen.	Ein **Hase hat** den **Bär** zum Feind.
HB8	Besitzt ein Verb eine Subjektbeziehung zu einem Klassennamen und weiterhin eine Akkusativobjekt-Beziehung zu einem weiteren Klassennamen, besteht eine Assoziation zwischen diesen Klassen.	Eine **Ente kennt** den **Hasen**. (Siehe Abbildung 4.16)
HB9	Besitzt ein Hilfsverb zwei Subjektbeziehungen zu unterschiedlichen Klassennamen, so besteht eine Assoziation zwischen diesen Klassen (andernfalls würden durch den Textgenerator zwei Sätze erstellt werden).	**Zauberer können Waffe** besitzen. (Siehe Abbildung 4.17)
HB10	Besitzt ein Verb zwei Subjektbeziehungen zu unterschiedlichen Klassennamen, so besteht eine Assoziation zwischen diesen Klassen (andernfalls würden durch den Textgenerator zwei Sätze erstellt werden).	**Zauberer mögen Waffe** gern.
HB11	Besitzt ein Hilfsverb zwei Akkusativobjekt-Beziehungen zu unterschiedlichen Klassennamen, so besteht eine Assoziation zwischen diesen Klassen.	**Hase mögen Karotte** und Stroh.

HB12	Besitzt ein Verb zwei Akkusativobjekt-Beziehungen zu unterschiedlichen Klassennamen, so besteht eine Assoziation zwischen diesen Klassen.	**Hase kennen Ente**, Hund und Katze. (Siehe Abbildung 4.18)
HB13	Steht ein Klassenname in Form eines Genitivattributs zu einem weiteren Klassennamen, so besteht eine Assoziation zwischen diesen Klassen.	**Feind** des **Charakter** können Ork, Magier und Ritter sein. (Siehe Abbildung 4.19)
HB14	Besitzt ein Hilfsverb eine Subjektbeziehung zu einem Klassennamen und weiterhin eine Prädikat-Beziehung zu einem weiteren Klassennamen, so besteht eine Assoziation zwischen diesen Klassen.	**Feind** des Charakter **können Ork**, Magier und Ritter sein. (Siehe Abbildung 4.19)
HB15	Besitzt ein Verb eine Subjektbeziehung zu einem Klassennamen und weiterhin eine Prädikat-Beziehung zu einem weiteren Klassennamen, so besteht eine Assoziation zwischen diesen Klassen.	**Feind** des Charakter **brauchen Schwert**, Lanze und Speer als Waffe.
HB16	Besitzt ein Hilfsverb eine Subjektbeziehung zu einem Klassennamen und weiterhin eine Teilsatzbeziehung zu einem Verb, welches wiederum eine Akkusativobjekt-Beziehung zu einem weiteren Substantiv besitzt, so besteht eine Assoziation zwischen diesen Klassen.	[...] **kann Spieler** aus drei verschiedenen Charakter seine **Spielfigur wählen**. (Siehe Abbildung 4.20)

HB17	Besitzt ein Hilfsverb eine Modifikator-Beziehung zu einer Adposition, die wiederum eine Relation zu einem Klassennamen aufweist sowie eine Subjektbeziehung zu einem weiteren Klassennamen, so besteht eine Assoziation zwischen diesen Klassen.	[...] **kann Spieler aus** drei verschiedenen **Charakter** seine Spielfigur wählen. (Siehe Abbildung 4.20)
HB18	Besitzt ein Klassenname in einem Hauptsatz eine Relativsatz-Beziehung zu einem Hilfsverb in einem Nebensatz, welches wiederum eine Subjektbeziehung zu einem Klassennamen aufweist, so besteht eine Assoziation zwischen diesen Klassen.	[...] **Tasche** mitführen, in der verschiedene **Gegenstand** gesammelt werden **können**. (Siehe Abbildung 4.21)
HB19	Besitzt ein Hilfsverb eine Subjektbeziehung zu einem Klassennamen und einer numerischen Aufzählung, welche wiederum eine Genitivattribut-Beziehung zu einem weiteren Klassennamen hat, so besteht eine Assoziation zwischen den Klassen.	**Feind** können ebenfalls bis zu **zwei** der obigen **Waffe** besitzen. (Siehe Abbildung 4.22)
HB20	Ist ein oder sind mehrere Signalworte für Vererbungsbeziehungen in einem Satz beschrieben, wird zusätzlich untersucht, ob diese zwei Prädikat-Beziehungen zu Klassennamen aufweisen. Ist dies nicht der Fall, erfolgt keine Zuordnung der Vererbungsbeziehung (sofern HB6 oder HB21 nicht in Kraft treten).	Ein **Feind ist** ein **Charakter**. (Siehe Abbildung 4.23)

HB21	Ist ein oder sind mehrere Signalworte für Vererbungsbeziehungen in einem Satz beschrieben, wird zusätzlich untersucht, ob diese eine Prädikat-Beziehungen sowie eine Subjektbeziehung zu Klassennamen aufweisen. Ist dies nicht der Fall, erfolgt keine Zuordnung der Vererbungsbeziehung (sofern HB6 oder HB20 nicht in Kraft treten).	**Zauberer**, Fee und Zwerg **sind Charakter**. (Siehe Abbildung 4.24)

Tabelle 4.3: Regelwerk zur Extraktion von Beziehungen zwischen detektierten Klassen.

Die Erkennung von Beziehungen zwischen zwei oder mehr Klassen in einem Satz ist nur dann möglich, wenn dieser mehr als einen Klassennamen enthält. Diese Grundvoraussetzung ist in Regel HB1 festgehalten.

Die in Tabelle 4.1 angeführten Signalworte sind auch für die Identifikation von Beziehungen relevant. Stehen die Satzbausteine „ist ein“ zwischen zwei Klassennamen, impliziert dies eine Vererbungsbeziehung (siehe HB2). Auch für multiple als Klasse identifizierten Substantive können Vererbungsbeziehungen zu einer weiteren Klasse aufweisen. In diesem Fall findet das Signalwort „sind“ Anwendung, wie in Regel HB3 ersichtlich ist. Da das Liskov Substitution Principle (siehe Unterabschnitt 4.3.2) bei der Erstellung textueller Anforderungsbeschreibungen berücksichtigt wird, erfolgt bei HB2 und HB3 keine zusätzliche Prüfung für dieses Kriterium. Wie ihre Namen bereits implizieren, weisen die Signalworte „Aggregation“ und „Komposition“ auf Aggregations- respektive Kompositionsbeziehungen hin. Befindet sich ein oder mehrere Klassennamen vor dem jeweiligen Signalwort, handelt es sich um die Teilkomponenten in der Teil-Ganzes-Beziehung. Der auf das Signalwort folgende Klassenname deutet folglich das Ganze hin. Die Regeln HB4 und HB5 fassen dies zusammen. Stehen Substantive in einer Aufzählung hintereinander und ist über ein Signalwort eine Beziehung zu einem weiteren Nomen beschrieben, so gilt diese Relation für alle aufgezählten Klassen. Dies gilt auch für Beziehungen, die nicht mithilfe eines spezifischen

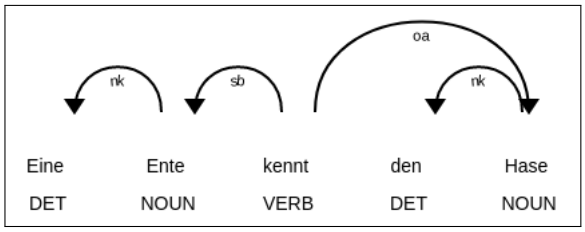


Abbildung 4.16: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem fünften Beispielsatz.

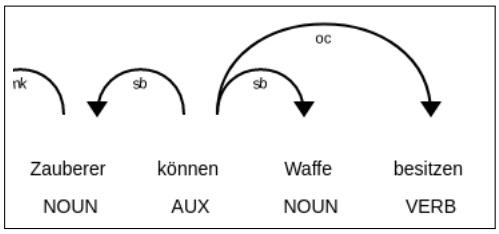


Abbildung 4.17: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem sechsten Beispielsatz.

Wortes detektierbar sind. Dies lässt sich in Regel HB6 nachlesen.

Sofern der Beziehungstyp nicht explizit über Signalworte beschrieben ist, sind die Relationen zwischen Klassen nur über die von spaCy detektierten Verbindungen der Satzbausteine erkennbar. Abbildung 4.16 verdeutlicht dies beispielhaft. An dieser Stelle sei noch einmal auf die Vorverarbeitung der einzelnen Sätze und die Überführung der Substantive in ihre Grundform hingewiesen. Besitzt ein Hilfsverb oder ein Verb eine Subjektbeziehung zu einem Nomen und eine Akkusativobjekt-Beziehung zu einem weiteren Substantiv, besteht eine Assoziation zwischen beiden Klassennamen (siehe HB7 und HB8). Gleiches gilt, wenn ein Hilfsverb oder ein Verb zwei Subjektbeziehungen zu Klassennamen aufweist, wie Abbildung 4.17 aufzeigt und HB9 sowie HB10 festhält. Diese Regeln sind auch anwendbar, wenn spaCy statt Subjektbeziehungen zwei Akkusativobjekt-Beziehungen detektiert, wie das Beispiel in Abbildung 4.18 aufzeigt. Nach HB11 und HB12 liegt folglich eine Assoziation zwischen den Klassen „Hase“ und „Ente“ vor. Ersterer wird aufgrund von HB6 auch eine Beziehung zu „Hund“ und „Katze“ hinzugefügt. Wie HB13 zusammenfasst, besteht weiterhin eine Relation zwischen zwei Substantiven, wenn eines als Genitivattribut zum anderen steht (von spaCy werden diese Beziehungstypen mit der Abkürzung „ag“ versehen). In Abbildung 4.19 lässt sich dies beispielhaft nachvollziehen. Weiterhin verdeutlicht

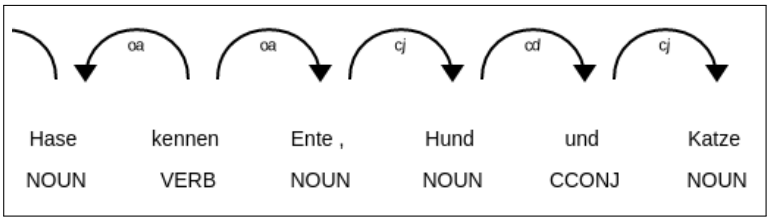


Abbildung 4.18: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem siebten Beispielsatz.

diese die Regeln HB14 und HB15, welche besagt, dass eine Assoziation zwischen zwei Klassennamen besteht, sofern ein Hilfsverb oder Verb eine Subjektbeziehung zu einem der beiden und eine Prädikat-Relation zum jeweils anderen aufweist. Wie die Satzanalyse in Abbildung 4.20 darstellt, besteht eine Assoziationsbeziehung, wenn ein Hilfsverb eine Subjektbeziehung zu einem Nomen und weiterhin eine Teilsatzbeziehung zu einem Verb, welches wiederum eine Akkusativobjekt-Beziehung zu einem weiteren Substantiv besitzt, aufweist (siehe HB16). Weiterhin kann in einem Relativsatz ein Nomen in einem Hauptsatz Bezug auf ein Hilfsverb oder Verb in einem Nebensatz nehmen. Zusätzlich visualisiert das Schaubild Regel HB17. Besitzt ein Hilfsverb eine Modifikator-Beziehung (bei spaCy mit „mo“ abgekürzt) zu einer Adposition und diese wiederum eine Beziehung mit einem Substantiv, besteht eine Assoziation, falls das Hilfsverb eine Subjektbeziehung zu einem weiteren Klassennamen hat. Abbildung 4.21 zeigt dies beispielhaft. Die im englischen als „relative clause“ (und von spaCy als „rc“) beschriebene Beziehung eines Nomens zu einem Hilfsverb ist dort einsehbar. Weist letzteres zusätzlich eine Subjektbeziehung zu einem Substantiv auf, besteht eine Assoziation zwischen beiden Klassennamen (siehe HB18). Die darauffolgende Regel ist anhand von Abbildung 4.22 visualisiert. Zu sehen ist ein Hilfsverb, das eine Subjektbeziehung zu einem Nomen und einer numerischen Aufzählung aufweist, welche wiederum eine Genitivattribut-Beziehung zu einem weiteren Substantiv besitzt. HB19 beschreibt diesen Sachverhalt.

Wie bereits erläutert, sind Vererbungsbeziehungen anhand beschriebener Signalworte erkennbar. Eine simple Zuweisung ist jedoch nicht sinnvoll, da weitere Klassen mit anderen Beziehungstypen in einem Satz beschrieben sein können. Wie Abbildung 4.23 und 4.24 verdeutlichen, erfolgt eine Untersuchung mithilfe von spaCy auf Prädikat-Beziehungen der Signalworte zu Substantiven (siehe HB20) oder einer Subjektbeziehung und einer Prädikat-

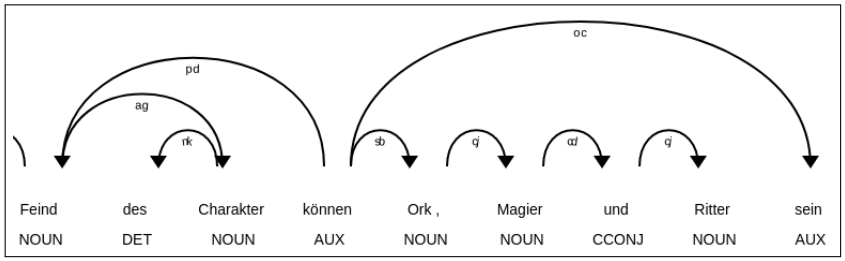


Abbildung 4.19: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem achten Beispielsatz.

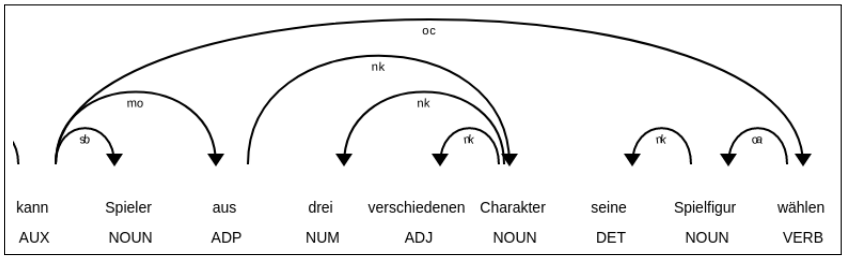


Abbildung 4.20: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem neunten Beispielsatz.

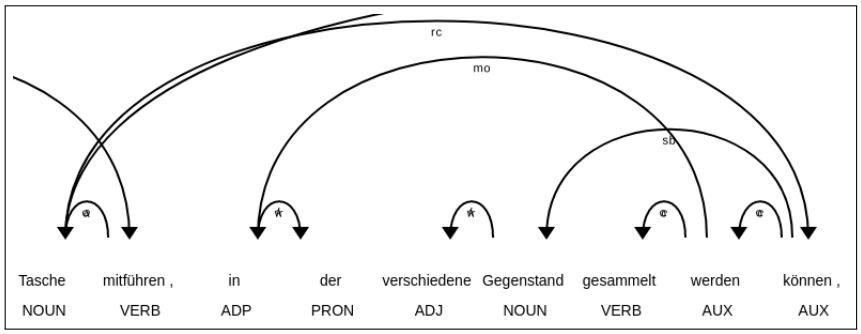


Abbildung 4.21: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem zehnten Beispielsatz.

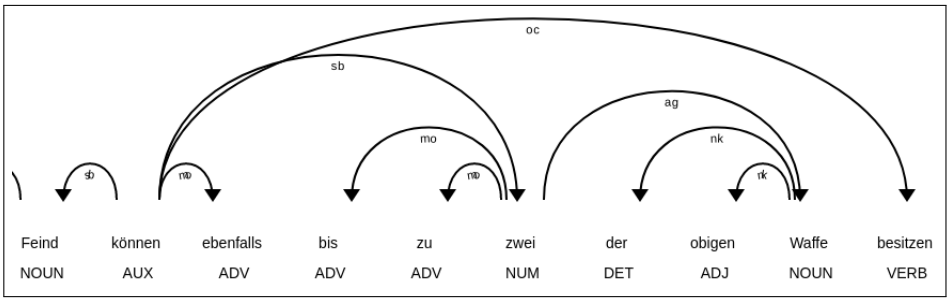


Abbildung 4.22: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem elften Beispielsatz.

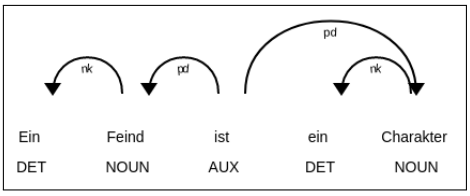


Abbildung 4.23: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem zwölften Beispielsatz.

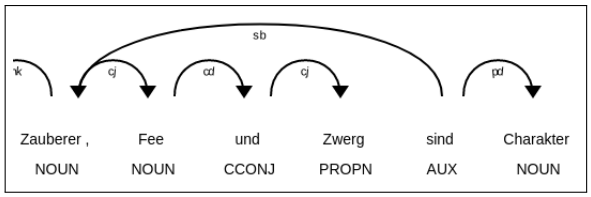


Abbildung 4.24: Erkennung von Worttypen und Wortrelationen mithilfe der spaCy-Bibliothek in einem dreizehnten Beispielsatz.

Beziehung der Signalworte (siehe HB21).

Das Entfernen von Duplikaten ist nach Anwendung der heuristischen Regeln der letzte Schritt bei der Extraktion von Beziehungen zwischen Klassen. Die Speicherung aller Klasseninformationen erfolgt in einer separaten Dictionary-Struktur.

4.4.2.3 Regelwerk für die Extraktion von Entwurfsmustern und die Erstellung zugehöriger Klassen und Beziehungen

Enthält ein Satz das Signalwort „Entwurfsmuster“, so ist eine gesonderte Betrachtung notwendig. Die bis dato aufgestellten Regeln finden keine Anwendung. Dieser Schritt muss der Extraktion von Klasseninformationen nachgelagert sein, damit eine Zuordnung bestehender und neu zu erstellender Klassen in Form von Beziehungen möglich ist.

Vorerst ist zu identifizieren, um welches Entwurfsmuster es sich handelt. Ein dem Signalwort vor- oder nachgelagertes Nomen ist dafür ausschlaggebend. Weiterhin kann der Satz Substantive enthalten, die für die Erstellung der Klassennamen innerhalb des Entwurfsmusters relevant sind. Außerdem können bestehende Klassennamen beschrieben sein, mit denen sich die zu erstellenden Klassen verknüpfen lassen.

Die konzipierte Vorgehensweise ist übertragbar und soll im Folgenden durch einen Beispielsatz beschrieben werden:

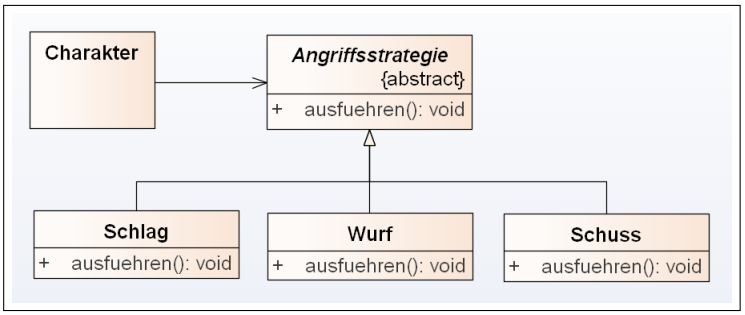


Abbildung 4.25: UML-Klassendiagramm für einen Beispielsatz, der das Strategie-Entwurfsmuster beschreibt.

Mithilfe des Strategie-Entwurfsmusters kann der Charakter zwischen verschiedenen Angriffsstratgien wie Schlag , Wurf oder Schuss wählen.

Darin beschrieben ist das Strategie-Entwurfsmuster, mit einer abstrakten Klasse „Angriffsstrategie“, den Kindklassen „Schlag“, „Wurf “ und „Schuss“ sowie einer Verknüpfung zu einer bestehenden Klasse „Charakter“. Abbildung 4.25 zeigt die Klassen mit ihren Relationen. Mithilfe von spaCy sind die einzelnen Klassennamen sowie deren Beziehungen zueinander erkennbar und extrahierbar. Die Ergebnisse werden in den bestehenden Dictionary-Strukturen hinterlegt und reflektieren das in der Abbildung dargestellte UML-Klassendiagramm.

4.4.3 Implementierung

Die in Codebeispiel 4.5 abgebildete Klasse ‚DiagramGenerator‘ stellt die Fassadeklasse des gleichnamigen Softwarepakets dar. Als Kommunikationsschnittstelle zwischen den jeweiligen Tasks sowie ggf. externen Softwarekomponenten nimmt sie eine textuelle Anforderungsbeschreibung entgegen und liefert eine Musterlösung in Form einer XML-Struktur zurück. Die in der Klasse enthaltenen Methode ‚generate_overview_about_detectable_-elements()‘ ermöglicht eine Markierung von UML-Klassendiagrammkomponenten mithilfe von HTML-Annotationen. In einem ersten Schritt erfolgt eine Vorverarbeitung des Texts durch die Klasse ‚Preprocessor‘. Codebeispiel 4.6 verdeutlicht, wie diese implementiert werden kann. Wie alle anderen Klassen innerhalb des Softwarepakets ‚Task‘, erbt sie von der gleichnami-

```
import spacy

from DiagramGenerator.Task.Preprocessor import Preprocessor
from DiagramGenerator.Task.ExtractClassInformation
                            import ExtractClassInformation
from DiagramGenerator.Task.FlagClassInformation
                            import FlagClassInformation
from DiagramGenerator.Task.ExtractClassRelations
                            import ExtractClassRelations
from DiagramGenerator.Task.ExtractDesignPattern
                            import ExtractDesignPattern
from DiagramGenerator.Task.CreateXML import CreateXML

class DiagramGenerator:

  def generate_overview_about_detectable_elements(self):

    preprocessor = Preprocessor(self.__exercise_text)
    sentences = preprocessor.run()

    flag_class_information = FlagClassInformation(sentences,
                                        self.__nlp,
                                        self.__non_objective_words,
                                        self.__non_method_verbs)
  res = flag_class_information.run()

  return res

  def generate_sample_solution(self):

    preprocessor = Preprocessor(self.__exercise_text)
    sentences = preprocessor.run()

    sentences = self.remove_pattern_sentences(sentences)

    extract_class_information = ExtractClassInformation(
                                    sentences, self.__nlp,
                                    self.__non_objective_words,
                                    self.__non_method_verbs)
    class_information, cleared_class_information =
                                  extract_class_information.run()

    extract_class_relations = ExtractClassRelations(
                                    sentences, self.__nlp,
                                    self.__non_objective_words,
                                    self.__non_method_verbs,
                                    class_information,
                                    cleared_class_information)
    class_relations = extract_class_relations.run()

    extract_design_patterns = ExtractDesignPattern(
                                    self.__design_pattern_sentences,
                                    self.__nlp,
                                    cleared_class_information,
                                    class_relations)
    cleared_class_information, class_relations =
                                  extract_design_patterns.run()

    create_xml = CreateXML(cleared_class_information,
                                                class_relations)
    sample_solution_str = create_xml.run()

    return sample_solution_str
```

Codebeispiel 4.5: Beispielhafte Implementierung der Klasse ‚DiagramGenerator‘.

```
from DiagramGenerator.Task.Task import Task

class Preprocessor(Task):

 def run(self):

  sentences = self.split_sentences()
  sentences = self.replace_pronouns_with_nouns(sentences)
  sentences = self.lemmatize_words(sentences)

  return sentences
```

Codebeispiel 4.6: Beispielhafte Implementierung der Klasse ‚Preprocessor‘.

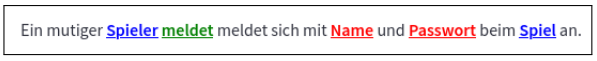


Abbildung 4.26: Beispielhafte Darstellung einer HTML-Annotation für einen Satz einer textuellen Anforderungsbeschreibung.

gen abstrakten Klasse und besitzt somit eine ‚run()‘-Methode, die durch die Fassadeklasse aufrufbar ist. Wie aus Codebeispiel 4.6 entnommen werden kann, unterteilt die Vorverarbeitung textuelle Anforderungsbeschreibungen in ihre individuellen Sätze, ersetzt Pronomen durch Nomen und führt letztlich eine Lemmatisierung von Substantiven durch. Als Rückgabewert liefert die ‚run()‘-Methode ein Array mit den individuellen Sätzen (als String). Im Anschluss erfolgt eine Instanziierung der Klasse ‚FlagClassInformation‘ (siehe Codebeispiel 4.7) und ein Methodenaufruf der darin enthaltenen ‚run()‘-Methode. Diese wiederum ruft die Methode ‚extract()‘ auf. Satzweise findet eine Analyse mithilfe der spaCy-Bibliothek statt. Daraufhin wird über die einzelnen Wörter eines Satzes iteriert. Das Codebeispiel zeigt exemplarisch das Vorgehen, falls ein Satzbaustein als Nomen klassifiziert wurde. Je nachdem, ob es sich bei dem Substantiv um ein gegenständliches (Klassenname) oder nichtgegenständliches (Attribut) handelt, erfolgt eine HTML-Annotation in blauer (für Erstere) respektive roter (für Letztere) Farbe. Der String ‚exercise_text‘ speichert alle Worte mit oder ohne Annotation. Er spiegelt gleichzeitig den Rückgabewert der Methode wieder. Enthalten ist die zuvor bereitgestellte, textuelle Anforderungsbeschreibung, die nun farbliche Markierungen beinhaltet, sofern eine Einbettung in eine Webseite erfolgt. Abbildung 4.26 zeigt beispielhaft das Ergebnis des Prozesses.

Die in der Fassadeklasse und in Codebeispiel 4.5 enthaltene Methode ‚generate_sample_solution()‘ dient der Erzeugung von Musterlösungen im XML-Format anhand textueller Anforderungsbeschreibungen. Wie zuvor

```
from DiagramGenerator.Task.Task import Task

class FlagClassInformation(Task):

 def run(self):

  sentences = self.extract()

  return sentences

 def extract(self):

  ...
  exercise_text = ""
  for sentence in self.sentences:
   doc = self.__nlp(sentence)
   for i in range(len(doc)):
    token = doc[i]
    if token.pos_ == "NOUN":
     if token.text in self.__non_objective_words:
      exercise_text += "<span style='color:red'><b><u>"
                       + token.text + "</u></b></span>"
     else:
      exercise_text += "<span style='color:blue'><b><u>"
                       + token.text + "</u></b></span>"

  ...

  return exercise_text
```

Codebeispiel 4.7: Beispielhafte Implementierung der Klasse ‚FlagClassInformation‘.

verarbeitet die Klasse ‚Preprocessor‘ diese vor. Anschließend erfolgt eine Entfernung von Sätzen, die Entwurfsmuster enthalten. Nun sind die UML-Klassendiagrammkomponenten extrahierbar. Hierfür erstellt die Methode eine Instanz der Klasse ‚ExtractClassInformation‘, welche das Regelwerk (siehe Tabelle 4.2) realisiert. Exemplarisch ist die Umsetzung von Regel HK12 (in verkürzter Form) aufgezeigt. Innerhalb der ‚run()‘-Methode erfolgt ein Aufruf der ‚extract()‘-Methode. Diese iteriert über die jeweiligen Sätze und die darin enthaltenen Worte. Ist einer dieser Satzbausteine von spaCy als Verb deklariert, erfolgt eine Prüfung auf eine Subjektbeziehung zu einem Substantiv. Ist Regel HK12 erfüllt, kann eine interne Zuordnung durchgeführt werden. Nach Fertigstellung des Extraktionsprozesses ist eine Entfernung möglicher Duplikate durchführbar. Abbildung 4.27 zeigt das Ergebnis anhand eines Beispielsatzes. Die Klassen ‚MutigerSpieler‘ (siehe Regel HK21 in Tabelle 4.2) und ‚Spieler‘ sind identifiziert worden. Ersterer sind die Attribute ‚Name‘ und ‚Passwort‘ sowie die Methode ‚anmelden‘ zugewiesen. Da keinerlei Sichtbarkeiten für diese angegeben sind, werden

```
from DiagramGenerator.Task.Task import Task

class ExtractClassInformation(Task):

 def run(self):

  class_information, class_information_list = self.extract()
  cleared_class_information = self.remove_duplicates(
                                        class_information_list)

  return class_information, cleared_class_information

 def extract(self):

  ...

  for sentence in self.sentences:
   doc = self.__nlp(sentence)
   ...
   elif token.pos_ == "VERB":
    for child in token.children:
     if child.pos_ == "NOUN" and child.dep_ == "sb":
     components_belonging_class = child.text
     ...
...

return class_information, class_information_list
```

Codebeispiel 4.8: Beispielhafte Implementierung der Klasse ‚ExtractClassInformation‘.

```
Ein mutiger Spieler meldet sich mit Name und Passwort beim Spiel an

{'Type': 'class', 'ID': 'Spieler', 'Name': 'MutigerSpieler', 'BelongsToClassWithID': None, 'Abstract': False, 'Attri
butes': ['-name', '-passwort'], 'Methods': ['+anmelden']}
{'Type': 'class', 'ID': 'Spiel', 'Name': 'Spiel', 'BelongsToClassWithID': None, 'Abstract': False, 'Attributes': [],
'Methods': []}
```

Abbildung 4.27: Beispielhafte Extraktion von Informationen auf Klassenbasis anhand des beschriebenen Regelwerks.

Attribute als ‚privat‘ und Methoden als ‚public‘ deklariert.

Der Code für die Extraktion von Beziehungen zwischen identifizierten Klassen ist vergleichbar strukturiert, wie Codebeispiel 4.9 prägnant aufzeigt. Die eigentliche Differenz besteht in der Realisierung der in Tabelle 4.3 angeführten heuristischen Regeln für die Identifikation von Beziehungen. Abbildung 4.28 verdeutlicht das Ergebnis dieses Vorgangs anhand eines Beispielsatzes. Für die Klassen ‚Spieler‘ und ‚Spiel‘ wurde eine Assoziationsbeziehung erkannt.

Durch die Fassadeklasse erfolgt anschließend ein Aufruf der Klasse ‚ExtractDesignPattern‘. Enthalten die Sätze keine Beschreibungen zu Entwurfsmustern, gibt sie die ihr übergebenen Informationen zu den jeweiligen Klassen respektive deren Beziehungen unbearbeitet zurück. Andernfalls wird

```
from DiagramGenerator.Task.Task import Task

class ExtractClassRelations(Task):

 def run(self):

  class_relations = self.extract()
  class_relations = self.remove_duplicates(class_relations)
  ...

  return class_relations

 def extract(self):

  ...

  return class_relations
```

Codebeispiel 4.9: Beispielhafte Implementierung der Klasse ‚ExtractClassRelations‘.

```
Ein mutiger Spieler meldet sich mit Name und Passwort beim Spiel an

{'From': ['Spieler'], 'To': ['Spiel'], 'Type': 'association'}
```

Abbildung 4.28: Beispielhafte Extraktion von Beziehungen detektierter Klassen anhand des beschriebenen Regelwerks.

eine Detektion des Entwurfsmusters sowie der damit zusammenhängenden Klassen durchgeführt. Als Rückgabewert dienen die erweiterten Klassen- bzw. Beziehungsinformationen. Die Implementierung berücksichtigt das Adapter-, Strategie- und Dekorierer-Entwurfsmuster (vgl. Eilebrecht und Starke, 2019). Der Code ist an dieser Stelle beliebig erweiterbar. Aufgrund der Vielfalt unterschiedlicher Entwurfsmuster ist eine Realisierung aller im Kontext dieser Arbeit nicht möglich.

In einem letzten Schritt ist der Export aller Informationen in eine XML-Datei möglich. Wie Codebeispiel 4.11 verdeutlicht, generiert der Code eine XML-Struktur und speichert diese in einer Datei. Diese ist beispielsweise mit der Modellierungsanwendung Enterprise Architect einlesbar und visualisierbar.

4.4.4 Evaluation

Für die Evaluation der automatisiert generierten UML-Klassendiagramme können die durch die vorangegangene Evaluation dieses Kapitels erstellen, textuellen Anforderungsbeschreibungen zugrunde gelegt werden. Die Überprüfung der Vollständigkeit und Korrektheit der Diagramme steht dabei im

```
from DiagramGenerator.Task.Task import Task

class ExtractDesignPattern(Task):

 def run(self):

  ...

  elif "Strategie" in sentence:
        self.__cleared_class_information, self.__class_relations =
             self.create_strategie_pattern_classes(sentence_nouns,
                                  self.__cleared_class_information,
                                  self.__class_relations)
    ...
  ...

  return self.__cleared_class_information, self.__class_relations
```

Codebeispiel 4.10: Beispielhafte Implementierung der Klasse ‚ExtractDesignPattern‘.

```
import xml

from DiagramGenerator.Task.Task import Task

class CreateXML(Task):

 def run(self):

  ...

  for i in range(len(classes)):
   new_element = xmldoc.createElement("element")
   new_element.setAttribute("xmi:type", "uml:Class")
   ...
  ...

  f = open(new_file, "w")
  f.write(xml)
  f.close()

  return pretty_xml
```

Codebeispiel 4.11: Beispielhafte Implementierung der Klasse ‚CreateXML‘.

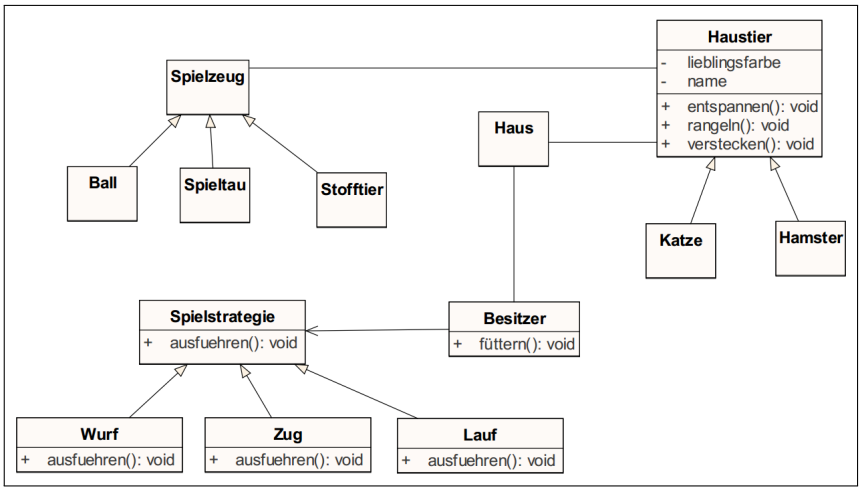


Abbildung 4.29: Im Kontext der Evaluation exemplarische und automatisiert erzeugte Musterlösung für eine gegebene Anforderungsbeschreibung.

Fokus.

Der Aufruf der Fassadeklasse des Softwarepakets ‚DiagramGenerator' erfolgt auch hier durch eine temporär generierte Klasse mit ‚main()'-Methode. Die Texte wurden dabei manuell eingefügt und der Fassadeklasse nacheinander übergeben. Eine Überprüfung seitens des Autors wäre anhand der resultierenden XML-Datei nur mit erheblichem Aufwand durchführbar. Daher erfolgte eine grafische Darstellung dieser mithilfe der Modellierungssoftware Enterprise Architect.

Abbildung 4.29 zeigt beispielhaft das erstellte UML-Klassendiagramm für die in Abbildung 4.6 dargestellte, textuelle Anforderungsbeschreibung. Alle enthaltenen Klassen sowie deren Attribute, Methoden und Beziehungen untereinander sind vollständig und korrekt abgebildet. Die übersichtliche Anordnung der Diagrammkomponenten erfolgte nach dem Import der XML-Datei in die Modellierungssoftware durch den Autor. Wie die Abbildung verdeutlicht, enthält das UML-Klassendiagramm keine Fehler. Das gilt auch für die neun weiteren, exemplarischen Resultate.

Die entwickelten Regelwerke liefern aufgrund der Satzstrukturen und beschriebenen Signalworte innerhalb der textuellen Anforderungsbeschreibung zuverlässige Ergebnisse. Die Abstimmung beider Softwarekomponenten aufeinander ist erforderlich. Andernfalls wäre eine umfassende, linguistische Forschung in diesem Kontext notwendig, die den Rahmen dieser Dissertation sprengt. Weiterhin ist die vorgestellte Konzeption mit seinen Re-

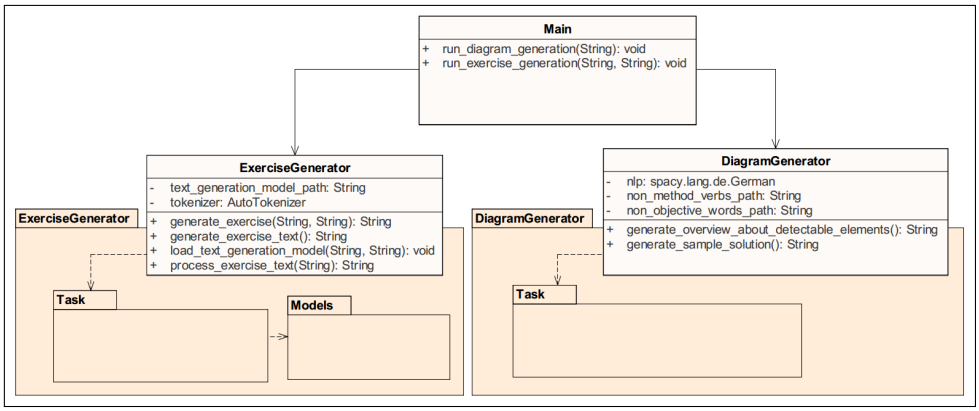


Abbildung 4.30: Zusammenführung der UML-Klassendiagramme für die automatisierte Generierung textueller Anforderungsbeschreibungen und zugehörigen Musterlösungen.

gelwerken beliebig erweiterbar und könnte auch als Grundlage für weitere Aufgabentypen dienen. Neben diesen angeführten Vorteilen gibt es auch Herausforderungen, die sich aufgrund des gewählten Ansatzes ergeben. Die Konzeption geht davon aus, dass Beschränkungen für die Fachtexte gelten. So wird beispielsweise angenommen, dass in einem Satz genannte Klassennamen auch eine Relation zueinander besitzen. Sind diese Beschränkungen nicht berücksichtigt, können die produzierten UML-Klassendiagramme unvollständig oder fehlerhaft sein. Weiterhin sind einige Konzepte explizit zu beschreiben. Exemplarisch können hier die Aggregation und die Komposition angeführt werden. Ohne explizite Definition dieser in den textuellen Anforderungsbeschreibungen gäbe es mehrere mögliche Lösungsmöglichkeiten, die jedoch nicht alle fachlich korrekt sein müssen (in diesen Fällen wären in der Modellierungspraxis entsprechende Stakeholder zu konsultieren, was im hochschulischen Übungsbetrieb nachvollziehbarer Weise nicht möglich ist). Die explizite Beschreibung und ein Ausschluss der Verwendung von Synonymen (eine Erklärung hierzu folgt im nächsten Kapitel) schränken den Lösungsraum ein. Wie studentische UML-Klassendiagramme mit Musterlösungen verglichen werden und welche Überlegungen dabei berücksichtigt sind, ist ebenfalls im Folgenden erläutert.

4.5 Zusammenführung

Der Aufbau der Softwarepakete für die automatisierte Generierung tex-

```
from ExerciseGenerator.ExerciseGenerator import ExerciseGenerator
from DiagramGenerator.DiagramGenerator import DiagramGenerator

class Main:

 def run_exercise_generation(self, difficulty, theme):

  exercise_generator = ExerciseGenerator()
  generated_exercise = exercise_generator.generate_exercise(
                                          difficulty, theme)

 return generated_exercise

 def run_diagram_generation(self, exercise_text):

  diagram_generator = DiagramGenerator()
  sample_solution_str
        = diagram_generator.generate_sample_solution(
                                        exercise_text)

  return sample_solution_str
```

Codebeispiel 4.12: Beispielhafte Implementierung einer Klasse für die Zusammenführung der automatisierten Erstellung von textuellen Anforderungsbeschreibungen sowie zugehörigen Musterlösungen.

tueller Anforderungsbeschreibungen und dazugehörigen Musterlösungen ist modular (siehe UML-Klassendiagramme in Abbildung 4.5 und 4.11). Einzelne Softwarebausteine sind somit beliebig erweiterbar oder austauschbar. Durch Einsatz des Fassade-Entwurfsmusters können jedoch auch die Softwarepakete selbst modular in eine Umgebung eingebettet werden. Die Fassadeklassen fungieren als Kommunikationsschnittstelle zwischen Softwarekomponenten innerhalb und außerhalb der Softwarepakete. Für die Verknüpfung dieser Bausteine ist lediglich eine weitere Klasse nötig, wie das UML-Klassendiagramm in Abbildung 4.30 zeigt. Die in Schaubild 4.5 und 4.11 dargestellten Pakete sind aus Gründen der Übersichtlichkeit in verkürzter Form enthalten. Über eine zusätzliche Klasse (hier mit dem Namen ‚Main') ist eine Instanziierung beider Fassadeklassen mit anschließenden Funktionsaufrufen realisierbar (siehe Codebeispiel 4.12). Sie separiert dies in die Methoden ‚run_exercise_generation(...)' und ‚run_diagram_generation(...)'. Erstere liefert nach Übergabe eines gewählten Schwierigkeitsgrads und eines Themenschwerpunkts eine automatisiert generierte Anforderungsbeschreibung. Diese dient anschließend als Eingabeparameter für die Erstellung einer zugehörigen Musterlösung. Die Ergebnisse beider Schritte sind in den vorangegangenen Evaluationen einsehbar.

Um den vollen, implementierten Funktionsumfang zur Verfügung stellen

zu können, ist die Konzeption sowie Bereitstellung einer interaktiven Weboberfläche notwendig. So sind die in Form von HTML-Annotationen generierten, farblichen Markierungen, die in den Anforderungsbeschreibungen enthaltene UML-Klassendiagrammkomponenten farblich hervorhebt, darstellbar. Für die Kommunikation zwischen Weboberfläche und den beschriebenen Softwarepaketen ist die Implementierung einer Server-Klasse sinnvoll. Diese kann Anfragen respektive Nutzereingaben entgegennehmen und Ergebnisse bereitstellen. In der Konzeption einer softwaregestützten Hilfestellung für die hochschulische Lehre von UML-Klassendiagrammen ist dies zu berücksichtigen.

4.6 Beantwortung von Forschungsfrage 1.1

Dieses Kapitel gab Aufschluss darüber, wie eine prototypische Software für die Generierung von Übungsaufgaben sowie zugehöriger Musterlösungen konzipiert und implementiert werden kann. Der bereitgestellte Funktionsumfang orientiert sich dabei an den in Kapitel 3 erhobenen Anforderungen.

Neben einer Untersuchung verwandter Arbeiten erfolgte eine Diskussion geleisteter Vorarbeiten. Diese dienten als Ausgangsbasis für die Konzeptionen und das jeweilige Softwaredesign in Form von UML-Klassendiagrammen. Weiterhin wurden mögliche Implementierungen für diese beschrieben. Nach den Evaluationsschritten erfolgte eine Zusammenführung beider Softwarekomponenten.

Die angeführten Ergebnisse beantworten, wie eine Software zur Generierung textueller Aufgaben zu UML-Klassendiagrammen sowie zugehöriger Musterlösungen konzipiert und prototypisch implementiert werden kann (Forschungsfrage 1.1) hinreichend.

KAPITEL 5

Vergleich studentischer Lösungen mit automatisch generierten Musterlösungen

Die Abstraktion objektorientierter Konzepte zu einem vollständigen, den Anforderungen entsprechendem Softwaredesign ist für Studierende keine einfache Aufgabe (vgl. Saini u. a., 2019; vgl. Wei u. a., 2016). Der Vergleich einer studentischen Lösung mit einer vorhandenen Musterlösung ist für alle Akteure zeitaufwendig, da sich die Diagramme voneinander unterscheiden und dennoch korrekte Lösungen darstellen können (vgl. Schots u. a., 2010; vgl. Huber und Hagel, 2020; vgl. Huber und Hagel, 2022b). Dies gilt nicht nur für die Anordnung der Komponenten, sondern auch deren Inhalte. So wäre beispielsweise, je nach Kontext und Aufgabenstellung, die Verwendung des Wortes „Fahrzeug“ anstelle des Wortes „Auto“ ebenfalls korrekt. Studierenden ist somit oftmals auch nach der Präsentation einer Musterlösung durch Dozierende unklar, wo genau die Differenzen beider Diagramme bestehen, ob es sich bei ihrem UML-Klassendiagramm um eine korrekte Alternativlösung handelt und wie es zu den ggf. vorhandenen Fehlern kam.

Ähnlich wie bei den Lernstrategien, unterscheidet sich auch das Vorgehen bei der Erstellung des Softwaredesigns unter Studierenden stark voneinander (vgl. Stikkolorum u. a., 2015). Während teils spezifische Modellierungsanwendungen präferiert werden, finden in der Lehrpraxis auch Papier und Stift nach wie vor Anwendung (vgl. Huber und Hagel, 2020). Aus

Abschnitt 3.1 geht hervor, dass bei Tools zur Unterstützung der hochschulischen Lehre von UML-Klassendiagrammen jedoch meist eine spezifische Modellierungsanwendung ausgewählt und eingesetzt wird. Solche sind in ihrem Funktionsumfang oftmals zu komplex für den Lehrkontext und tragen mehr zu allgemeinen Verwirrung, als zum Lernerfolg bei (vgl. Krusche u. a., 2020).

Das Forschungsdesiderat der Frage 1.2 (siehe Abschnitt 1.1) befasst sich daher mit der Konzeption und prototypischen Implementierung eines Systems für den Vergleich studentischer UML-Klassendiagramme mit automatisch generierten Musterlösungen. Es soll ein von der Erstellungsvariante unabhängiges Einlesen studentischer Lösungen ermöglichen und Differenzen beider Diagramme identifizieren. Studierende können somit auf konkrete Fehler in ihrem Softwaredesign hingewiesen werden und diese in Zukunft vermeiden. Es ist weiterhin herauszuarbeiten, wie eine Ähnlichkeit zwischen zwei Diagrammen identifizierbar ist und bis zu welchem Grad überprüft werden kann, ob ein studentisches Klassendiagramm trotz Differenzen zu einer Musterlösung ebenfalls eine korrekte Lösung darstellt. Alle Untersuchungen stehen dabei in Relation zu der zuvor definierten, inhaltlichen Struktur der Übungsaufgaben.

5.1 Vorgehen und Einordnung in das Forschungsdesign

Nach den Vorgaben des DSR-Forschungsdesigns von Hevner u. a. (2004) bestehen realweltliche Herausforderungen, die durch den vorherigen Abschnitt erörtert und mithilfe eines zu entwickelten Artefakts anzugehen sind. Aus der Umgebung resultierende Anforderungen (siehe Kapitel 3) dienen auch hier als Entwicklungsgrundlage. Die in den Kapiteln 2 und 3 beschriebene Wissensbasis wird in Abschnitt 5.2 mithilfe einer Untersuchung zu Forschungsfrage 1.2 verwandter Arbeiten erweitert. Für die Konzeption (siehe Abschnitt 5.3.3) und prototypische Implementierung (siehe Abschnitt 5.3.4) dient diese als Basis. Für eine Bewertung des Artefakts und zur Identifikation von Optimierungspotenzialen wird das entstandene Artefakt anhand ausgewählter Methodiken in der Umgebung evaluiert (siehe Abschnitt 5.3.5).

Abschließend kann eine Antwort auf Forschungsfrage 1.2 in Abschnitt 5.4 dargelegt werden.

5.2 Verwandte Arbeiten (Related Work)

Durch eine softwaregestützte Hilfestellung könnte Abhilfe in Bezug auf nicht verfügbares individuelles und zeitnahes Feedback für Studierende durch Lehrpersonen geschaffen werden (vgl. Huber und Hagel, 2022b). Wie die Autoren in ihrer systematischen Literaturübersicht anführen, ist der Vergleich studentischer Lösungen mit Musterlösungen, einem definierten Set von Regeln oder vergleichbaren Strukturen das am häufigsten gewählte Vorgehen bei softwaregestützten Hilfestellungen in diesem Kontext. Deren Funktionalitäten sind in Kapitel 3 ausführlich festgehalten und finden Einzug in die folgende Konzeption. Letztere ist demnach eine Symbiose und Erweiterungen der in der Literaturübersicht aufgelisteten Softwareansätze. Weiterhin ist zu berücksichtigen, dass keine spezifischen Softwaresysteme vorausgesetzt werden, die nicht allen Hochschulen und Universitäten zur Verfügung stehen. Durch das visuelle Einlesen von Klassendiagrammen ließe sich beispielsweise auf vorbestimmte Modellierungsanwendungen verzichten.

Für die Informationsextraktion aus Bilddateien von UML-Klassendiagrammen gibt es eine Publikation von Chen u. a. (2022). Der von den Autoren vorgestellte Ansatz ist in der Lage, Klassen und Interfaces sowie Beziehungskomponenten zu klassifizieren und ggf. deren Texte zu extrahieren (vgl. Chen u. a., 2022). Er wurde nicht speziell für die hochschulische Lehre entwickelt und unterscheidet sich in seiner Herangehensweise bei der Detektion der Diagrammbestandteile von der bei Huber und Hagel (2020) präsentierten. Lediglich Bilddateien und keine XML-Dateien sind einlesbar. Auch der Vergleich einer studentischen Lösung mit einer Musterlösung findet hier keine Berücksichtigung. Weiterhin ist sie auf die Erkennung englischer Worte spezialisiert.

Alle durch den Autor identifizierten, verwandten Arbeiten lassen sich gegenüber der folgenden Konzeption abgrenzen.

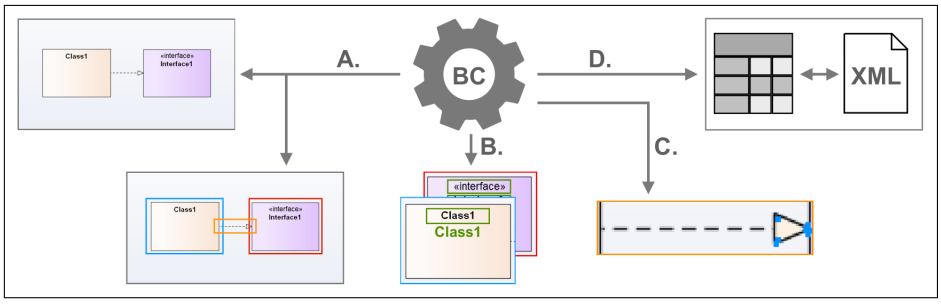


Abbildung 5.1: Architektur eines Prototyps für das Detektieren von Komponenten eines UML-Klassendiagramms sowie deren Vergleich mit einer vorhandenen Musterlösung (Huber und Hagel, 2020, S. 4).

5.3 Vergleich zweier UML-Klassendiagramme

5.3.1 Vorarbeit

Durch den Autor dieser Arbeit (neben Weiteren) wurde bereits eine Work-In-Progress Publikation zu diesem Thema veröffentlicht (vgl. Huber und Hagel, 2020). Dabei handelte es sich um einen bis dato nicht evaluierten Prototyp, der unter Zuhilfenahme von Deep-Learning-Technologien Komponenten eines UML-Klassendiagramms detektieren, analysieren und die extrahierten Informationen mit einer Musterlösung vergleichen kann. Dieser nimmt studentische UML-Klassendiagramme in Form von Bildern entgegen und vergleicht diese mit einer durch Dozierende erstellte Musterlösung im XML-Format. Da es sich um eine reine Konsolenanwendung (ohne grafische Benutzeroberfläche) handelt, gibt der Prototyp erkannte Unterschiede beider Diagramme textuell aus.

Bei der Modellierung haben Studierende unterschiedliche Herangehensweisen. So nutzen sie beispielsweise diverse Softwareprogramme oder zeichnen ihre Diagramme von Hand. Weiterhin erlauben nicht alle eingesetzten Modellierungsanwendungen den Export einer Datei im XML-Format. Um die Studierenden nicht einzuschränken, nimmt der Prototyp von Huber und Hagel (2020) deren Modellierungen (ausschließlich) in Form eines Bildes entgegen. In folgenden Weiterentwicklungsschritten (die jedoch weder bei den genannten Autoren noch in dieser Arbeit im Fokus stehen) wäre durch diese Verfahrensweise auch das Einlesen handschriftlich erstellter UML-Klassendiagramme möglich.

Die in einem bereitgestellten Bild abgebildeten Diagrammkomponenten

werden mithilfe des in Abbildung 5.1 dargestellten Verfahrens ausgewertet und letztlich mit einer vorhandenen Musterlösung verglichen. Die in der Abbildung einsehbaren Schritte erfolgen, unter Verwendung eines Batch-Sequenziell Architekturstils (vgl. Starke, 2014), sequenziell. Hierbei ist es üblich, dass die aufeinanderfolgenden Verarbeitungsschritte durch eine zentrale Komponente gesteuert werden. Bei Huber und Hagel (2020) ist diese als „Batch-Controller“ oder kurz „BC“ bezeichnet. Die jeweiligen Schritte sind im Folgenden detailliert beschrieben. Um Konsistenz mit der Nummerierung innerhalb der Abbildung herzustellen, sind diese von A. bis D. sortiert (eigene Übersetzung und Ausführung in Anlehnung an (Huber und Hagel, 2020, S. 4f.)):

A. **Import eines Bildes sowie Detektion und Lokalisation von UML-Klassendiagrammkomponenten:** In einem ersten Schritt erfolgt der Import eines Bildes. Für die Python-Programmiersprache gibt es unterschiedliche, installierbare Bibliotheken, die das Einlesen solcher Dateien mit nur einer Codezeile ermöglichen. Eine merklich größere Herausforderung stellt die Lokalisation, Detektion und Klassifikation der auf dem importierten Bild abgebildeten Objekte dar. Herkömmliche Methoden wie beispielsweise Kantendetektoren wären in der Lage, abgebildete Elemente zu erkennen und deren Position innerhalb des Bildes zu bestimmen. Um eine Aussage über den Typ der UML-Klassendiagrammkomponenten treffen zu können, sind diese Ansätze jedoch ungeeignet. Für diese Aufgaben wurde daher das in Unterunterabschnitt 2.2.4.3 vorgestellte YOLO-Modell in der zum Zeitpunkt der Veröffentlichung aktuellsten, dritten Generation herangezogen. Für das Training sind 1000 selbst erstellte Bilder und Label verwendet worden. Da es sich um einen Work-In-Progress handelt, ist keine Erkennungsrate auf ungesehenen Bildern spezifiziert. Jedoch geben die Autoren an, dass in den ersten Tests nicht alle enthaltenen UML-Diagrammkomponenten detektiert werden konnten.

B. **Texterkennung:** Diagrammkomponenten wie Klassen, Interfaces und Enumerations enthalten üblicherweise diverse Texte, die beispielsweise Klassennamen oder Attribute spezifizieren. Für die Lokalisation von Textregionen wurde der sogenannte Efficient and Accuracy Sce-

ne Text (EAST) Detektor (vgl. Zhou u. a., 2017) verwendet. Zum Einlesen einzelner Textbausteine der jeweiligen Textregionen kam die open-source Bibliothek Tesseract[27] zum Einsatz. Nach diesem Verarbeitungsschritt sind die in einer studentischen Lösung enthaltenen Objekte lokalisiert, klassifiziert und deren textuellen Inhalte (falls vorhanden) importiert.

C. **Detektion der Richtung einer Beziehung:** Bis dato ist bekannt, wo sich Beziehungskomponenten befinden und um welche es sich handelt. Unbekannt ist jedoch, welche sonstigen Diagrammkomponenten diese miteinander verbinden und in welche Richtung sie weisen. Um diese Information zu erhalten, können die von YOLO erzeugten Bounding Boxen von Beziehungskomponenten und sonstigen Diagrammkomponenten auf Überschneidungen geprüft werden. In Abbildung 5.1 ist dies bei Punkt A. mit einer blauen, orangenen und roten Bounding-Box ersichtlich. Sofern die Beziehung nicht als ungerichtete Assoziation klassifiziert ist, befindet sich an einem Ende ein Symbol (in Abbildung 5.1 ein geschlossenes Dreieck). Dieses ist beispielsweise mit einem Harris Corner Detector (vgl. Harris und Stephens, 1988)[28] identifizierbar. Innerhalb des Dreiecks, um das Beispiel der Abbildung fortzuführen, können Ecken erkannt und durch Koordinaten ausgewiesen werden. Durch die Positionierung der Koordinaten ist ersichtlich, an welcher sonstigen Diagrammkomponente ein richtunggebendes Symbol einer Beziehungskomponente anliegt.

D. **Vergleich:** Alle bisherigen Informationen sind in Arrays gespeichert und abrufbar. In einem Pandas DataFrame[29] werden diese nun zusammengeführt und in tabellarischer Struktur gespeichert. Professionelle Modellierungssoftware erlaubt oftmals den Export von UML-Klassendiagrammen in XML-Dateien. Diese sind durch den Prototyp importierbar und auswertbar. Extrahierte Informationen werden ebenfalls in einem DataFrame gespeichert und ermöglicht so einen Vergleich mit

[27] Beispielsweise einsehbar unter: https://opensource.google/projects/tesseract oder ttps://github.com/tesseract-ocr/tesseract

[28] Codebeispiel unter: https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

[29] Dokumentation einsehbar unter: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

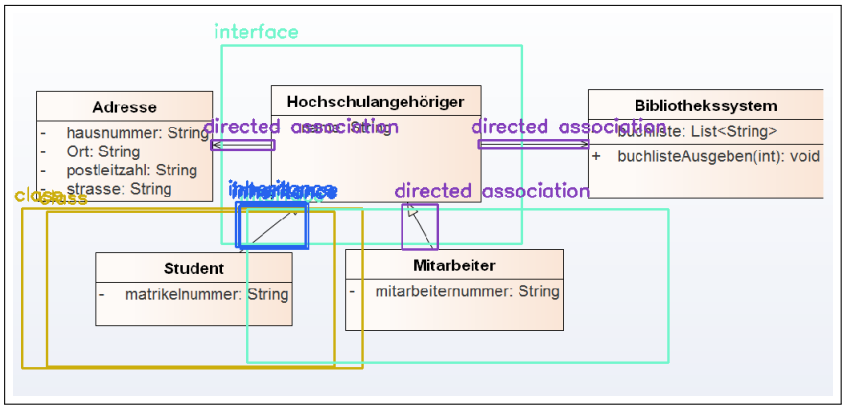


Abbildung 5.2: Erkennung von UML-Klassendiagrammkomponenten mithilfe des von Huber und Hagel (2020) vorgestellten YOLO in dritter Generation.

den Informationen der studentischen Lösung.

Wie Huber und Hagel (2020) beschreiben, ist der Prototyp in der hochschulischen Lehre von UML-Klassendiagrammen auf dem präsentierten Entwicklungsstand nicht einsetzbar. Gründe hierfür sind, dass Diagrammkomponenten durch das verwendete YOLO nicht detektiert oder Texte nicht vollständig / korrekt ausgelesen werden. Abbildung 5.2 verdeutlicht diese Problematik. Darauf sichtbar ist ein durch den Autor dieser Arbeit beispielhaft erstelltes UML-Klassendiagramm. Sichtbar sind fünf Klassen, zwei gerichtete Assoziationen (im oberen Bereich des Bildes) und zwei Vererbungen (im unteren Bereich des Bildes). Es ist ersichtlich, dass die enthaltenen Objekte teils falsch, mehrfach oder überhaupt nicht erkannt werden. Mit anderen Beispielen ließe sich zeigen, dass die Erkennung der Diagrammbestandteile durchaus auch vollständig und korrekt funktioniert. Abbildung 5.2 soll speziell die erläuterte Problematik aufzeigen.

5.3.2 Ähnlichkeit zweier UML-Klassendiagramme

Ziel dieses Kapitels ist die Entwicklung einer Verfahrensweise, mit deren Hilfe automatisiert alle Differenzen zweier vorliegender UML-Klassendiagramme identifizierbar sind. Die von Huber und Hagel (2020) zugrundeliegende Designentscheidung, das visuelle Einlesen von Diagrammen zu ermöglichen, ist aufgrund der angeführten Argumente durchaus sinnvoll und soll daher auch in der vorliegenden Konzeption berücksichtigt werden. Darüber hin-

aus soll es allen beteiligten Akteuren möglich sein, UML-Klassendiagramme sowohl als Bild- als auch in Form einer XML-Datei bereitzustellen. Die angeführte Vorarbeit wies diese Flexibilität nicht auf. Trotz der Zuverlässigkeit der entwickelten Verfahrensweise kann es in bestimmten Situationen vorkommen, dass der visuelle Import ein nicht vollständig korrektes Ergebnis liefert (beispielsweise wenn die Auflösung des eingegebenen Bildes schlecht ist). Die Empfehlung liegt daher auf der Verwendung maschinell erstellter UML-Klassendiagramme im XML-Format. Alle Untersuchungen beschränken sich im Rahmen dieser Arbeit auf solche, die mithilfe der Modellierungssoftware Enterprise Architect erstellt wurden. Die professionelle Anwendung ermöglicht den Export beider Formate. Das Konzept soll jedoch erweiterbar sein, sodass zukünftig beispielsweise auch handschriftliche Lösungen importierbar sind.

Für eine Bewertung studentischer UML-Klassendiagramme ist die Bestimmung von Ähnlichkeit zu einer Musterlösung vorzunehmen. Dem Duden zufolge ist mit dem Adjektiv „ähnlich“ alles beschreibbar, was in bestimmten Merkmalen übereinstimmt (vgl. Cornelsen Verlag GmbH, 2023a). Bei einer vollkommenen Übereinstimmung ist hingegen das Wort „identisch“ zu wählen (vgl. Cornelsen Verlag GmbH, 2023b). Sind zwei Diagramme nicht vollkommen identisch, unterscheiden sie sich in mindestens einem Merkmal voneinander. Alle Abweichungen (im folgenden auch mit dem Substantiv „Fehler“ beschrieben) sind durch den Prototyp zu erfassen, zu benennen und zu speichern. Eine Bestimmung des Ähnlichkeitsgrads anhand eines Bewertungsschemas ist in Kapitel 6 vorgesehen, da es hier lediglich um die Identifikation aller Differenzen geht.

Die Übungsaufgaben sind durch das vorherige Kapitel so konzipiert, dass sie einer vordefinierten Struktur folgen. Sie schließen die Verwendung von Synonymen und zusätzlichen Diagrammbestandteilen bei Modellierungen explizit aus. Die Studierenden sollen sich ausschließlich an den gegebenen Texten orientieren. Ob eine davon abweichende Lösung korrekt sein kann, ist in den allermeisten Fällen nur durch kontextuelles Expertenwissen verifizierbar. Wie in der Einordnung in den Problemkontext dieses Kapitels angeführt, kann beispielsweise situativ die Verwendung des Klassennamens „Fahrzeug“ statt des Wortes „Auto“ korrekt sein. Als Personenkraftwagen ist das Auto jedoch von einem Lastkraftwagen zu unterscheiden, obwohl es

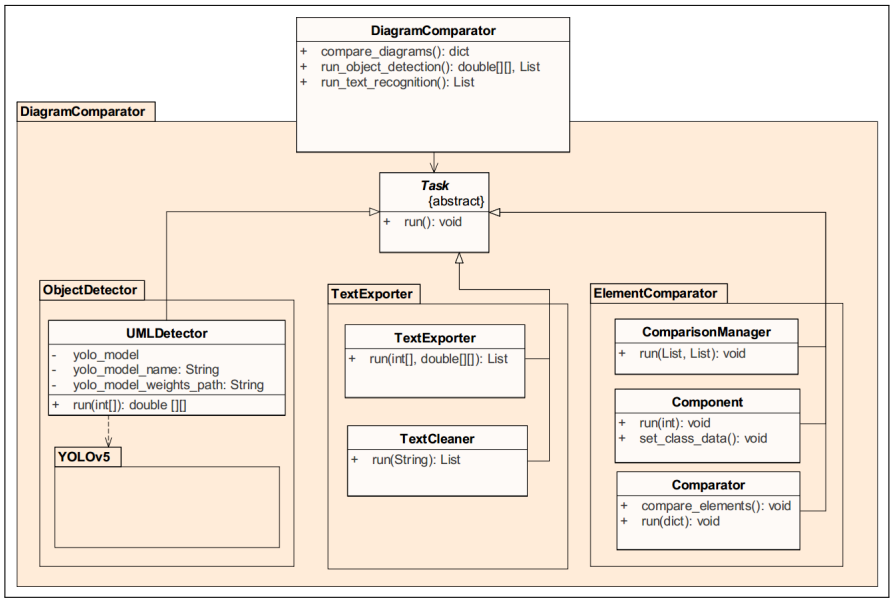


Abbildung 5.3: Softwaredesign der Softwarekomponente für die automatisierte Detektion von UML-Klassendiagrammkomponenten in Bildern sowie den Vergleich zweier UML-Klassendiagramme miteinander.

sich jeweils um ein Fahrzeug handelt. In einem Softwaresystem, in dem nur Autos vorhanden sind, könnte die Verwendung beider Substantive zulässig sein, obwohl sie eigentlich nicht als Synonyme gelten. Durch ein automatisiertes System sind solche kontextuellen Entscheidungen in vielen Fällen nicht zuverlässig möglich. Die Einschränkung der Fachtexte schließt solche Situationen aus und ermöglicht somit eine Bestimmung der Ähnlichkeit zweier UML-Klassendiagramme.

5.3.3 Konzeption

Dem zyklischen Vorgehen nach Hevner u. a. (2004) folgend soll auch hier die beschriebene Vorarbeit als Entwicklungsgrundlage herangezogen werden. Daraufhin erfolgt eine Erweiterung respektive Optimierung dieser unter Einbezug der zuvor genannten Aspekte.

Das in Abbildung 5.3 einsehbare UML-Klassendiagramm visualisiert das entwickelte Softwaredesign. Der Aufbau orientiert sich anhand des Fassade-Entwurfsmusters (vgl. Eilebrecht und Starke, 2019; vgl. Kleuker, 2018). Aus Gründen der Übersichtlichkeit sind nur die relevantesten Klassen, Attribute, Methoden und Relationen dargestellt.

Alle Klassen der Softwarekomponente befinden sich innerhalb des Pakets

‚DiagramComparator'. Eine gleichnamige Fassadeklassen dient als Kommunikationsschnittstelle zu den darin enthaltenen Unterpaketen. Sofern die darin enthaltenen Klassen Programmlogik beinhalten, erben sie von der Klasse ‚Task' und besitzen somit eine 'run()'-Methode. Dies ermöglicht eine Trennung von Zuständigkeiten (vgl. Vogel u. a., 2011) sowie Zuweisung eindeutiger Verantwortlichkeiten (vgl. Martin, 2018) innerhalb einzelner Klassen. Folglich entsteht eine modulare Struktur, welche eine einfache Erweiterung oder einen Austausch von Softwarebausteinen erlaubt. Der Hauptfokus des Pakets liegt auf dem Vergleich zweier UML-Klassendiagramme miteinander. Ziel ist die Extraktion von Differenzen beider Diagramme. In diesem Sinne erfolgt ein syntaktischer Vergleich. Eine semantische Überprüfung findet nicht statt. Dafür ist eine Überführung von Informationen aus UML-Klassendiagrammen in eine interne, vergleichbare Repräsentation notwendig. Wie die Anforderungen in Kapitel 3 aufzeigen, sind Klassendiagramme sowohl als Bilddatei, als auch als XML-Datei zu importieren. Die Softwarekomponente muss somit eine Informationsdetektion und -extraktion anhand von Bildern ermöglichen. Daher sind die einzelnen Tasks je nach Aufgabengebiet in die drei Unterpakete ‚ObjektDetector', ‚TextExporter' und ‚ElementComparator' gegliedert. Diese dienen (in dieser Reihenfolge) der Extraktion von UML-Klassendiagrammkomponenten aus Bildern (falls ein zu vergleichendes Diagramm nicht im XML-Format vorliegt), dem Auslesen der in den Klassen abgebildeten Texte (falls ein zu vergleichendes Diagramm nicht im XML-Format vorliegt) sowie dem Vergleich zweier UML-Klassendiagramme miteinander.

Das in Abbildung 5.3 dargestellte Unterpaket ‚ObjectDetector' beinhaltet folgende Komponenten:

- ‚UMLDetector': Die Klasse ruft den in Unterunterabschnitt 2.2.4.3 präsentierten Objektdetektor YOLOv5 auf und liefert ein zweidimensionales Array mit den Bounding-Box-Koordinaten der einzelnen Diagrammbestandteile sowie die Zuordnung detektierter Assoziationen zu den jeweiligen Klassen.

- ‚YOLOv5': Der Ordner enthält alle Dateien, die für die Ausführung von YOLOv5 notwendig sind.

In dem abgebildeten Unterpaket ‚TextExporter' sind die folgenden Klas-

sen enthalten:

- ‚TextExporter‘: Exportiert die in einer Klasse visuell dargestellten Texte zu einem String. Wie bei Huber und Hagel (2020) kommt dafür die open-source Bibliothek Tesseract zum Einsatz. Das Ergebnis stellt eine Liste mit extrahierten Information aus einer einzelnen Diagrammkomponente dar (sind mehrere in einem UML-Klassendiagramm enthalten, erfolgt eine iterative Informationsextraktion).

- ‚TextCleaner‘: Bereinigt die extrahierten Texte einer Diagrammkomponente von nicht korrekt identifizierten Symbolen. So kann es beispielsweise vorkommen, dass ein Gleichzeichen, statt eines Minuszeichens (welches auf eine die Sichtbarkeit Privat vor Attribut- oder Methodennamen hindeutet) durch Tesseract erkannt wurde. Die Nachverarbeitung der Textbausteine ist somit ein notwendiger Prozessschritt. Als Rückgabewert liefert die ‚run()‘-Methode der Klasse eine Liste mit Klasseninformationen.

Das letzte der drei Unterpakete mit Namen ‚ElementComparator‘ beinhaltet folgende Klassen:

- ‚ComparisonManager‘: Verwaltet die Überführung von Informationen aus UML-Klassendiagrammen (die entweder aus XML-Dateien oder aus Bildern stammen) in eine interne Repräsentation und steuert anschließend die Vergleichsoperationen zwischen zwei Repräsentationen.

- ‚Component‘: Bildet das Gerüst für die interne Repräsentation einer UML-Klasse mit ihren Attributen, Methoden und Beziehungen ab. Dabei spielt es keine Rolle, ob die Informationen aus einem Bild oder einer XML-Datei stammen. Durch eine einheitliche, interne Abbildung wird ein davon unabhängiger Vergleich zweier UML-Klassendiagramme möglich.

- ‚Comparator‘: Vergleicht zwei Instanzen der Klasse ‚Component‘ miteinander. Sie ist in der Lage, alle Differenzen zu detektieren und in einer Liste zu speichern.

Die angeführten Unterpakete stellen die Hauptbestandteile des konzipierten Softwarepakets dar, da sie die eigentliche Programmlogik enthalten.

In den folgenden Unterunterabschnitten sind deren Kernkomponenten daher näher erläutert. Sofern ein Bild und keine XML-Datei eingelesen werden soll, sind die einzelnen Diagrammbestandteile zu identifizieren, zu lokalisieren und deren Beziehungen untereinander zu bestimmen. Der hierfür ausgewählte Objektdetektor und der dafür durchgeführte Trainingsprozess sind in Unterunterabschnitt 5.3.3.2 erläutert. Eine zuverlässige Extraktion textueller Inhalte aus den einzelnen Diagrammbestandteilen ist ebenfalls relevant. In Unterunterabschnitt 5.3.3.3 ist dieser Prozess aufgeführt. Der letzte Unterunterabschnitt 5.3.3.4 beinhaltet eine Beschreibung zu den Vergleichen zweier UML-Klassendiagramme miteinander. In allen drei Fällen werden explizit die Unterschiede zu der Veröffentlichung von Huber und Hagel (2020) beleuchtet und somit aufgezeigt, wie das darin beschriebene Artefakt (vgl. Hevner u. a., 2004) konkret weiterentwickelt wurde.

5.3.3.1 Import von UML-Klassendiagrammen im XML-Format

Die Vorarbeit von Huber und Hagel (2020) fokussiert das Einlesen studentischer Lösungen in Form einer Bilddatei und das Einlesen von Musterlösungen in Form einer XML-Datei. Diese Restriktion soll durch die vorliegende Konzeption aufgehoben werden. Folglich muss es allen Akteuren möglich sein, sowohl Bilder als auch XML-Dateien zu erzeugten UML-Klassendiagrammen einzulesen. Wie in der Vorarbeit angeführt, weisen Studierende unterschiedliche Präferenzen bei Wahl des Modellierungswerkzeugs auf. Der Export von Inhalten ist in diesen nicht immer vorgesehen. Weiterhin soll das Einlesen handschriftlicher Klassendiagramme für zukünftige Entwicklungsschritte (die im Rahmen dieser Arbeit nicht enthalten sind) vorgesehen werden. All diese Aspekte sind von der Konzeption berücksichtigt. Der zu entwickelnde Prototyp muss daher in der Lage sein, diese unterschiedlichen Eingabeformate in eine interne Repräsentation zu überführen. Die Empfehlung liegt hierbei auf der Verwendung von XML-Dateien, da Fehler beim visuellen Einlesen nicht vollständig auszuschließen sind.

5.3.3.2 Detektion und Lokalisation der Diagrammbestandteile sowie deren Beziehungen untereinander

Wie Abbildung 5.2 verdeutlicht, genügt das von Huber und Hagel (2020) verwendete YOLOv3 aufgrund geringer Detektionsraten den Ansprüchen der hochschulischen Lehre von UML-Klassendiagrammen nicht. Untersuchungen zufolge besitzt das in Unterabschnitt 2.2.4.3 vorgestellte YOLOv5 eine merklich bessere Erkennungsrate (vgl. Nepal und Eslamiat, 2022). Es ist anzunehmen, dass der Einsatz dieses Objektdetektors zu besseren Ergebnissen bei der Erkennung von UML-Klassendiagrammkomponenten führt.

Um eine zuverlässige Detektion und Lokalisation von UML-Klassendiagrammkomponenten zu ermöglichen, wird daher in diesem Abschnitt das YOLOv5-Modell herangezogen. Als Basis für das Training dienen in einem ersten Schritt die von Huber und Hagel (2020) händisch erstellten 1000 Trainingsbilder mit zugehörigen Labeln. Es ist üblich, einen Teil des Datensatzes nicht für das Training, sondern für die spätere Evaluation zu verwenden. 200 Bilder (und zugehörige Label) sind daher dem Evaluationsdatensatz zugeordnet (20% des ursprünglichen Datensatzes). So kann eine stichhaltige Aussage über die Erkennungsrate des Modells auf bisher ungesehenen Bildern getroffen werden. Der Code für den Objektdetektor steht über GitHub zur Verfügung[30]. Weiterhin findet sich dort eine Anleitung[31] für das Training mit eigenem Trainingsdatensatz. Nach Abschluss des Trainingsvorgangs werden durch den Code automatisch statistische Auswertungen und Visualisierungen bereitgestellt. Sie dokumentieren sowohl die Inhalte der Datengrundlage, als auch die Erkennungsleistung und liefern daher wichtige Rückschlüsse für die Eignung des Modells für die hochschulische Lehre. Die Beschriftungen der im Folgenden angeführten Abbildungen sind automatisiert in englischer Sprache erzeugt worden.

Einen Einblick in den Inhalt des Trainingsdatensatzes liefert Abbildung 5.4. Dabei handelt es sich um eine Aneinanderreihung beispielhafter Bilder, die anhand der Informationen aus den Label-Dateien mit einer Bounding-Box und einer ID versehen sind. Letztere geben Aufschluss über den Typ

[30] Open-source Code für YOLOv5: https://github.com/ultralytics/yolov5/tree/v6.1

[31] Einsehbar unter: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

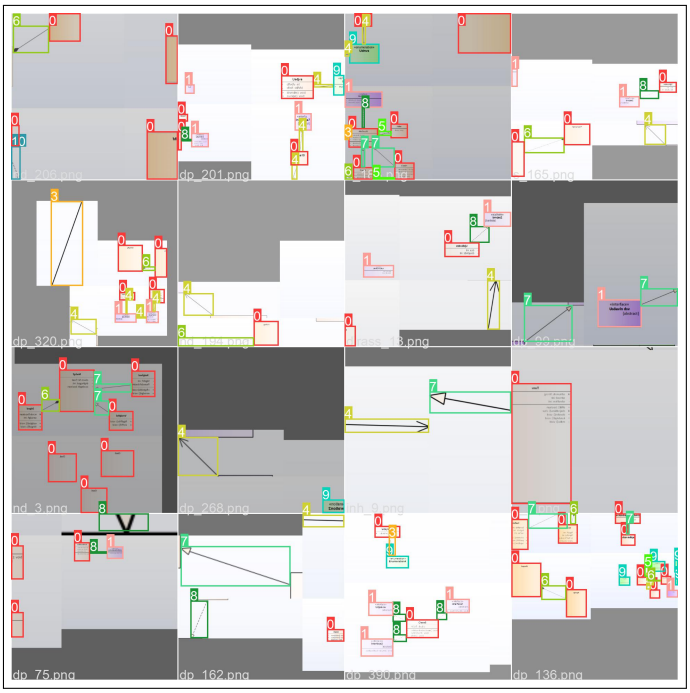


Abbildung 5.4: Beispielhafte Darstellung des von Huber und Hagel (2020) generierten Trainingsdatensatzes für das Training des YOLOv5-Modells.

der Diagrammkomponente. Eine Zuordnung der IDs zu den tatsächlichen Namen der Komponenten findet sich in Tabelle 5.1.

Von entscheidender Relevanz für die Erkennungsleistung ist die Häufigkeit, mit der die zu detektierenden Objekte in Summe in den einzelnen Bildern enthalten sind. Diese Verteilung ist in Abbildung 5.5 einsehbar. Die y-Achse beschreibt die Anzahl enthaltener Komponenten, während die x-Achse anhand der bereits erwähnten IDs bestimmt, um welche Diagrammkomponente es sich handelt. Die Übersicht zeigt, dass besonders Klassen in den Trainingsdaten stark vertreten sind. Für alle weiteren zu detektierenden Diagrammkomponenten sind deutlich weniger Beispiele vorhanden. Es ist anzumerken, dass Huber und Hagel (2020) den Typ *Abstrakte Klasse* (ID: 2) aus Gründen der Vollständigkeit hinzugefügt haben. Visuell sind diese jedoch in den meisten Modellierungsanwendungen lediglich durch das Signalwort „abstract“ oder einem in kursiv geschriebenen Klassennamen von herkömmlichen Klassen differenzierbar. Der zuverlässigere Ansatz ist daher, beide Komponenten mit dem Objektdetektor als Klasse zu erkennen. Bei der anschließenden inhaltlichen Auswertungen ist nachvollziehbar, ob es sich um eine abstrakte Klasse handelt. Für diesen Typ sind daher keine

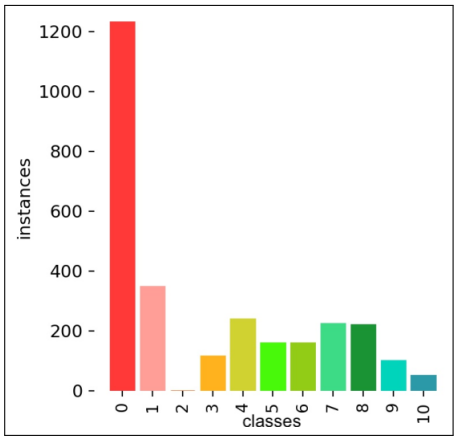


Abbildung 5.5: Verteilung innerhalb der von Huber und Hagel (2020) generierten Trainingsdaten.

ID	**Typ der Diagrammkomponente**
0	Klasse
1	Interface
2	Abstrakte Klasse
3	Assoziation
4	Gerichtete Assoziation
5	Aggregation
6	Komposition
7	Vererbung
8	Realisierung
9	Enumeration
10	Abhängigkeit

Tabelle 5.1: Zuordnung der von YOLOv5 vergebenen IDs zu dem jeweiligen Typ der UML-Klassendiagrammkomponenten.

Beispiele in den Trainingsdaten vorhanden. Dieser Ansatz soll auch in der vorliegenden Arbeit verfolgt werden.

Das Resultat des Trainingsvorgangs ist in Abbildung 5.6 dargestellt. Die auf der y-Achse abgebildete Präzision (engl. „Precision“) gibt an, wie viele der im Evaluationsdatensatz enthaltenen Objekte korrekt erkannt wurden. Der erreichte Grad der Zuversichtlichkeit (engl. „Confidence“) bei der Bestimmung der Typen dieser Objekte ist auf der x-Achse abgebildet. Besitzt das Modell eine hohe Präzision und Zuversichtlichkeit, ist davon auszugehen, dass es eine sehr gute Erkennungsleistung besitzt und somit zuverlässig eingesetzt werden kann. Wie Abbildung 5.6 verdeutlicht, trifft dies auf das

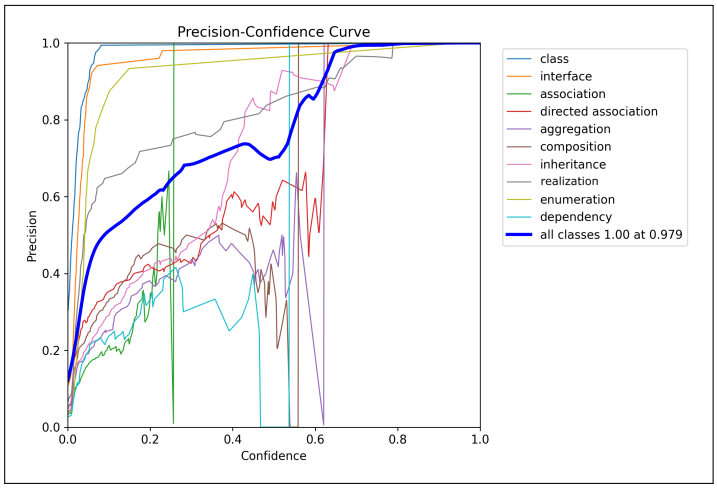


Abbildung 5.6: Trainingsergebnis des YOLOv5-Modells mit den von Huber und Hagel (2020) generierten Trainingsdaten.

trainierte Modell in seiner bisherigen Form nicht zu. Bei der Erkennung von Klassen, Interfaces und Enumerationen lassen sich gute Ergebnisse erzielen. Bei den meisten Beziehungskomponenten, die deutlich kleiner sind, sich einander sehr ähneln und keine Farbgebung aufweisen, sind die Werte für Präzision und Zuverlässigkeit jedoch ungenügend.

Eine naheliegende Schlussfolgerung ist, dass der vorhandene Trainingsdatensatz mit der Häufigkeit der enthaltenen Objekte und deren Diversität nicht ausreicht. Da die Erstellung von Trainingsdatensätzen mit großem Aufwand verbunden ist, wurden im Rahmen dieser Arbeit zwei Python-Skripte entwickelt, die diesen Prozess automatisieren. Die von Huber und Hagel (2020) erstellten Bilder sowie Label dienen weiterhin als Basis. Für deren Erweiterung schneidet das erste Skript die sichtbaren Objekte mehrerer zufällig ausgewählter Bilder aus. Mithilfe eines Konturdetektors [32] können die Umrisse von Diagrammkomponenten erkannt und anschließend als separate Bilder gespeichert werden. Da dieses Vorgehen keine Klassifikation des Typs der Diagrammkomponente ermöglicht, geben händisch erzeugte Dateinamen Aufschluss über das abgebildete Element. Das zweite Python-Skript ermöglicht die Erstellung von neuen Trainingsbildern und Labeln mit einer zufälligen Anzahl zuvor ausgeschnittener Diagrammkomponenten. Dafür gilt es in einem ersten Schritt, ein Hintergrundbild zu erzeugen.

[32] Ein Beispiel findet sich unter: https://learnopencv.com/contour-detection-using-opencv-python-c

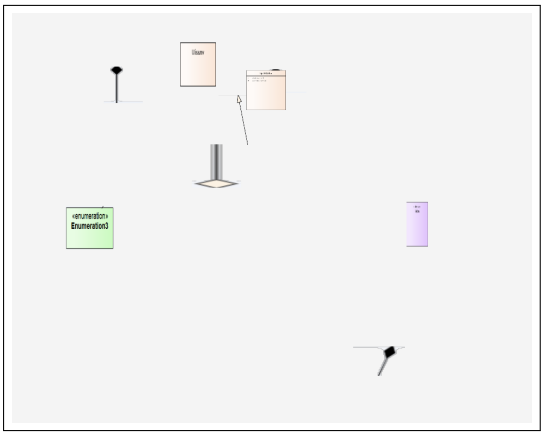


Abbildung 5.7: Beispielhafte Darstellung eines automatisiert erzeugten Traningsbildes für das YOLOv5-Modell.

Über randomisierte Zahlen variieren diese in Höhe und Breite sowie in der Farbgebung. Letztere kann unterschiedliche Weiß- und Grautöne annehmen. Anschließend findet eine zufällige Auswahl von drei bis zehn zuvor ausgeschnittenen Diagrammkomponenten statt. Sie werden mit randomisierten Zahlen skaliert und anschließend an beliebigen Positionen innerhalb des Bildes platziert. Die Beschränkung auf zehn Diagrammkomponenten erfolgte, da sich eine große Anzahl zufällig positionierter Objekte zu stark überlagern würde. Abbildung 5.7 zeigt eines der erzeugten Trainingsbilder beispielhaft. Da die Position sowie Höhe und Breite der abgebildeten Elemente innerhalb des Bildes bekannt sind, sind exakte Bounding-Boxen bestimmbar und in eine Label-Datei überführbar. 1000 weitere Trainingsbilder und Label wurden mithilfe des Skripts erzeugt. Gesamt sind folglich 2000 Trainingsbilder mitsamt Labeln verfügbar. Wie gehabt dienen 20% des Datensatzes als Evaluationsdatensatz.

Aufgrund unterschiedlicher Auswahl, Skalierung und Platzierung der Objekte ist von einer großen Diversität innerhalb des Datensatzes auszugehen. Abbildung 5.8 verdeutlicht, dass nun für jede Diagrammkomponente (abgesehen von abstrakten Klassen) mindestens 450 Beispiele vorhanden sind. Das Resultat des Trainings ist in Abbildung 5.9 dargestellt. Für alle Diagrammkomponenten ist eine Präzision und Grad der Zuversichtlichkeit über 90% erreicht. Im Kontext dieser Arbeit bedeutet dieses Ergebnis, dass sich das YOLOv5 sehr gut für die Detektion und Lokalisation von UML-Klassendiagrammkomponenten auf Bildern eignet. Beispielhaft ist dies in

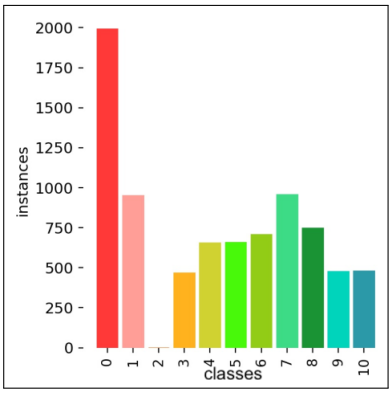


Abbildung 5.8: Verteilung innerhalb der von Huber und Hagel (2020) generierten sowie automatisiert erstellten Trainingsdaten.

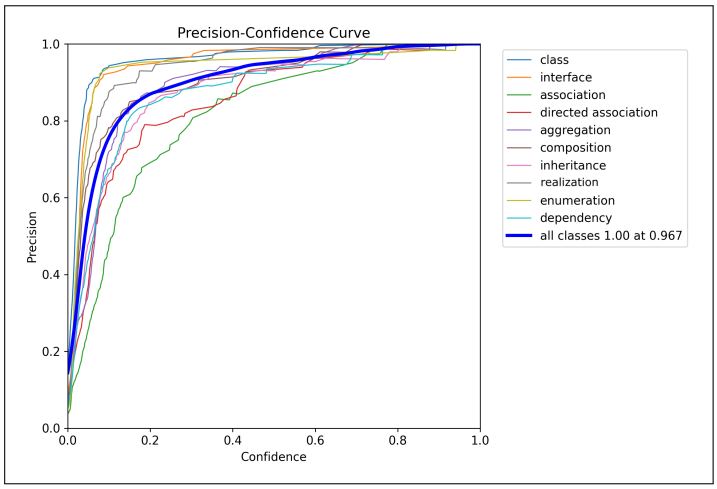


Abbildung 5.9: Trainingsergebnis des YOLOv5-Modells mit den von Huber und Hagel (2020) generierten sowie automatisiert erstellten Trainingsdaten.

Abbildung 5.10 verdeutlicht. Das für den Test des YOLOv3 erstellte Bild wird erneut herangezogen und dem YOLOv5 vorgelegt. Letzteres ist in der Lage, alle abgebildeten Diagrammkomponenten zu detektieren, lokalisieren und korrekt zu klassifizieren.

Das in der Konzeption verwendete YOLOv5 gehörte zum Zeitpunkt der Entwicklung der Softwarekomponente zu den aktuellen Modellen seiner Gattung. Mittlerweile gibt es Neuentwicklungen, die jedoch keinen Einzug in diese Dissertation finden. Das Training und das Hinzufügen eines neueren Objektdetektors ist aufgrund des modularen Aufbaus der Softwarekomponente möglich. Da mithilfe des verwendeten Modells enorm gute Ergebnisse erzielbar sind, ist jedoch fraglich, ob ein Austausch die Erkennungsrate merklich optimieren könnte. Da es sich bei der Modellfamilie um

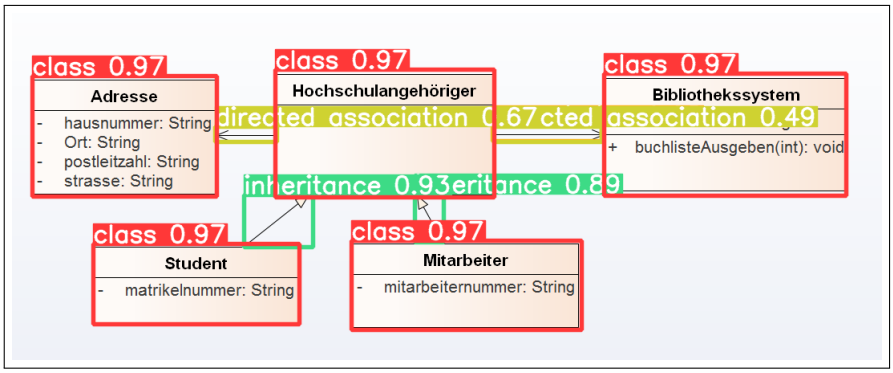


Abbildung 5.10: Erkennung von UML-Klassendiagrammkomponenten mit einem eigens trainierten YOLO in fünfter Generation.

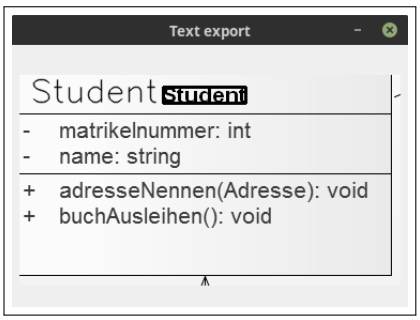


Abbildung 5.11: Extraktion von Texten mithilfe der open-source Bibliothek Tesseract.

Echtzeitdetektoren handelt, sind UML-Klassendiagrammkomponenten auf einem Kamerabild live erkennbar. Diese Echtzeiterkennung ist mithilfe der Softwarekomponente möglich, liegt jedoch nicht im Fokus der Arbeit und wird daher nicht explizit implementiert. Durch eine neuere YOLO-Version ließe sich diese ggf. performanter gestalten. Bei einer perspektivischen Weiterentwicklung wäre ein Austausch von YOLOv5 möglich. Für den aktuellen Entwicklungsstand ist die erzielte Detektionsrate jedoch vollkommen ausreichend.

5.3.3.3 Extraktion von Texten

Die von Huber und Hagel (2020) verwendete open-source Bibliothek Tesseract soll auch hier Anwendung finden. Sie ist in der Lage, Textregionen selbstständig zu identifizieren und auszulesen. Eine zusätzliche Lokalisation dieser Bildbereiche durch den von Huber und Hagel (2020) beschriebenen EAST-Detektor ist folglich substituierbar. Abbildung 5.11 zeigt beispielhaft

das Ergebnis einer analysierten Textregion. Um den Klassennamen ‚Student' befindet sich ein Rahmen, der durch den Detektor als Textregion identifiziert wurde. Links daneben steht der von Tesseract ausgelesene Text.

5.3.3.4 Vergleich zweier UML-Klassendiagramme

Der von Huber und Hagel (2020) beschriebene Prototyp nimmt Bilder studentischer UML-Klassendiagramme entgegen und vergleicht diese mit einer Musterlösung im XML-Format. Sowohl für Studierende als auch für Dozierende stand keine Auswahlmöglichkeit zwischen diesen Eingabeformaten zur Auswahl. In der vorliegenden Konzeption ist dieser Aspekt berücksichtigt. Beide Akteure sind in der Lage Bilder und XML-Dateien zu importieren.

Da die Autoren des Prototyps eine Konsolenanwendung entwickelt haben, ist lediglich eine textuelle Ausgabe der erkannten Differenzen möglich. Diese wurden bis dato jedoch nicht für weitere Bearbeitungsschritte gespeichert. Für ein umfassendes Feedback ist dies allerdings notwendig und ist daher in der Konzeption enthalten.

Für Klassen-, Attribut- und Methodennamen erfolgt vor der Instanziierung einer ‚Component'-Klasse mit diesen Informationen eine Rechtschreibprüfung. Die Softwarekomponente ist somit robust gegenüber erkennbaren Tippfehlern.

Weitere Optimierungen in diesem Bereich sind nicht erforderlich. Der von Huber und Hagel (2020) präsentierte Ansatz ermöglicht bereits einen vollumfänglichen Vergleich aller verwendbaren UML-Klassendiagrammkomponenten.

5.3.4 Implementierung

Aufgrund der großen Anzahl an Klassen und Operationen, die aus Gründen der Übersichtlichkeit in Abbildung 5.3 nicht vollständig darstellbar sind, sollen im Folgenden lediglich Codebeispiele der wichtigsten Komponenten angeführt werden. Diese beinhalten Auszüge aus der Fassadeklasse sowie der Klassen, welche die relevantesten Änderungen gegenüber dem Prototyp von Huber und Hagel (2020) einbeziehen.

Das Codebeispiel 5.1 zeigt eine beispielhafte Implementierung der Fassadeklasse. Sofern der Klasse eine oder mehrere Bilddateien übergeben wur-

```
from DiagramComparator.ObjectDetector.UMLDetector
                                    import UMLDetector
from DiagramComparator.TextExporter.TextExporter
                                    import TextExporter
from DiagramComparator.ElementComparator.ComparisonManager
                                    import ComparisonManager

class DiagramComparator:

 def compare_diagrams(self, student_diagram,
                      student_diagram_file_type,
                      solution, solution_file_type):

  comparison_manager = ComparisonManager()
  ...
  result = comparison_manager.execute_comparison(
                          student_diagram, solution,
                          student_diagram_file_type,
                                  solution_file_type)

  return result


 def run_object_detection(self, image):

  ...
  uml = UMLDetector()
  coordinates = uml.run_object_detection(image)
  associations = uml.get_association_intersections(
                                    coordinates, img_cp)

  return coordinates, associations


 def run_text_recognition(self, image, coordinates):

  text_exporter = TextExporter()
  class_information = text_exporter.export_text(
                                image, coordinates)

  return class_information
```

Codebeispiel 5.1: Beispielhafte Implementierung der Klasse ‚DiagramComparator‘.

```
import torch

class UMLDetector:

 def __init__(self):
  ...
  self.__yolov5_model = torch.hub.load('ultralytics/yolov5',
              'custom', path=self.path_yolov5_model_weights)


 def run_object_detection(self, image):

  results = self.__yolov5_model(image)

  return results.xyxy[0]
```

Codebeispiel 5.2: Beispielhafte Implementierung der Klasse ‚UMLDetector‘.

den, ruft sie die Methoden ‚run_object_detection(...)‘ und ‚run_text_-recognition(...)‘ auf. Erstere erstellt eine Instanz der Klasse ‚UMLDetector‘ und nutzt diese für die Detektion, Lokalisation und Klassifikation der UML-Klassendiagrammkomponenten. Als Rückgabewert liefert die Methode ein zweidimensionales Array mit Bounding-Box-Koordinaten und zugehörigen Identifikationsnummern für den erkannten Objekttyp sowie eine Liste detektierter Assoziationen mit ihren jeweiligen Zuordnungen zu einzelnen Klassen. Letztere Methode instanziiert ein Objekt der Klasse ‚TextExporter‘ und übergibt dessen Operation ‚export_text(...)‘ sowohl das Bild des Klassendiagramms, als auch die detektierten Bounding-Box-Koordinaten, was ein elementweises Auslesen von Texten erlaubt. Die Methode gibt eine Liste mit exportierten Texten als Ergebnis zurück. Die dritte in Codebeispiel 5.1 angeführte Methode ‚compare_diagrams(...)‘ erstellt, nach Übergabe zweier UML-Klassendiagramme sowie deren Dateitypen, eine Instanz der Klasse ‚ComparisonManager‘ und startet über einen Aufruf die Vergleichsoperationen. Je nach Dateityp ist anhand der aus einer Bild- oder XML-Datei extrahierten Informationen ein Objekt der Klasse ‚Component‘ für jedes zu vergleichende Diagramm zu erstellen. Das Codebeispiel geht darauf nicht explizit ein. Der Rückgabewert von ‚compare_diagrams(...)‘ ist eine Dictionary-Struktur, welche die Differenzen beider Klassendiagramme beinhaltet.

Das Codebeispiel 5.2 zeigt exemplarisch die Implementierung der Klasse ‚UMLDetector‘. Der darin abgebildete Konstruktor (in Python durch eine ‚__init__()‘-Methode realisiert) lädt das YOLOv5-Modell mit den vorab

```
import pytesseract

class TextExporter:

 def export_text(self, image, coordinates):

  for bb in coordinates:
   ...
   text = pytesseract.image_to_string(tmp_image,
                lang="deu", config=custom_config)
   ...
   class_information.append(text)

  return class_information
```

Codebeispiel 5.3: Beispielhafte Implementierung der Klasse ‚TextExporter‘.

```
from DiagramComparator.ElementComparator.Comparator
import Comparator

class ComparisonManager:

def execute_comparison(self, student_diagram, solution,
 student_diagram_file_type, solution_file_type):

 ...
 element_comparator = Comparator()
 element_comparator.compare_elements(self.components)
 result = element_comparator.print_comparison()

 return result
```

Codebeispiel 5.4: Beispielhafte Implementierung der Klasse ‚Comparison-Manager‘.

trainierten Gewichten. Wie Methode ‚run_object_detection(...)‘ aufzeigt, kann das Objekt des Detektors mit einem zu verarbeitenden Bild aufgerufen werden. Als Rückgabewert liefert die Methode ein zweidimensionales Array mit Bounding-Box-Koordinaten und einer eindeutigen Identifikationsnummer für die jeweilig erkannten Objekttypen.

Die Inhalte der Klasse TextExporter sind (in gekürzter Form) in Codebeispiel 5.3 einsehbar. Nach Übergabe des Bildes und aller detektierter Bounding-Box-Koordinaten iteriert die Methode ‚export_text(...)‘ über alle lokalisierten Objekte und prüft, ob es sich um auswertbare UML-Klassendiagrammkomponenten handelt (für Beziehungen zwischen beispielsweise Klassen wird folglich keine Textanalyse durchgeführt). Sofern dies zutrifft, erfolgt eine Extraktion mithilfe von Tesseract. Die Resultate sind in einer Liste speicherbar, die gleichzeitig auch als Rückgabetyp der Methode dient.

In der Methode ‚execute_comparison(...)‘ der Klasse ‚ComparisonMana-

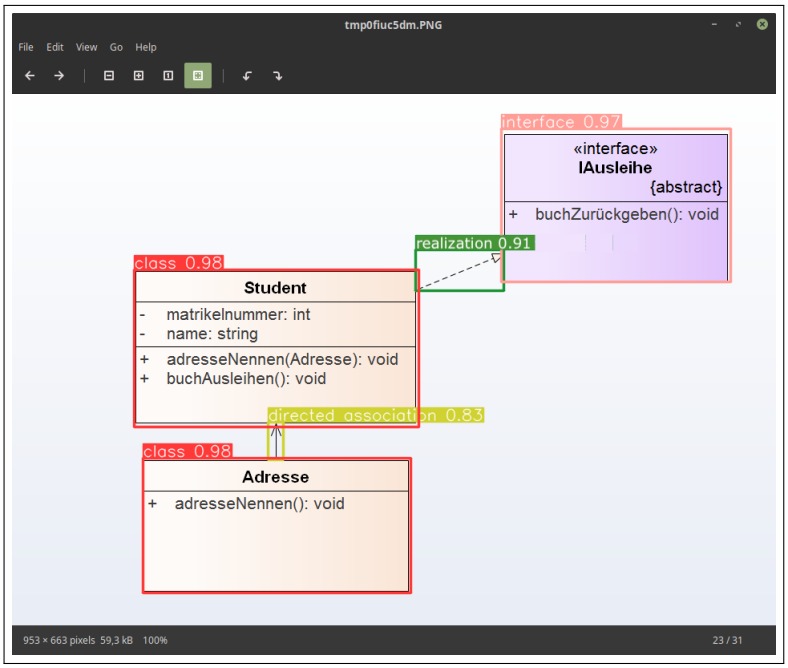


Abbildung 5.12: Exemplarische Detektion, Lokalisation und Klassifikation von UML-Klassendiagrammkomponenten für eine Evaluation der konzipierten Softwarekomponente.

ger‘ wird eine Instanz der Klasse ‚Comparator‘ erstellt. Durch einen Funktionsaufruf ist diese nach Übergabe zweier ‚Component‘-Objekte in der Lage, die Felder der internen Repräsentationen miteinander zu vergleichen und alle Differenzen in einer Dictionary-Struktur zu speichern. Letztere stellt auch den Rückgabewert der Methode ‚execute_comparison(...)‘ dar. Dies ist in Codebeispiel 5.4 dargestellt.

5.3.5 Evaluation

Für die Evaluation der Softwarekomponente ist sowohl eine Prüfung der visuellen Detektion und Informationsextraktion als auch eine Auswertung des Vergleichs zweier UML-Klassendiagramme notwendig.

Für Erstere sind durch den Autor beispielhaft fünf UML-Klassendiagramme mithilfe der Modellierungssoftware Enterprise Architect erstellt worden. Anschließend erfolgte eine manuelle Prüfung der von YOLOv5 bestimmten Typen aller Diagrammbestandteile sowie der extrahierten Texte. Abbildung 5.12 beinhaltet exemplarisch eines der Diagramme. Semantische Zusammenhänge innerhalb der abgebildeten Komponenten sind für die Eva-

luation nicht von Relevanz. In allen fünf Beispielen ist der Objektdetektor in der Lage, die enthaltenen UML-Klassendiagrammbestandteile vollständig zu erkennen und korrekt zu klassifizieren. Die Resultate lassen in Kombination mit den vorgestellten Trainingsergebnissen darauf schließen, dass YOLOv5 für den definierten Anwendungsfall sehr gut geeignet ist. Die Auswertung einzelner Textbausteine kann hingegen noch Herausforderungen darstellen. Die Resultate für den Textexport aus dem dargestellten Klassendiagramm finden sich in Form einer Konsolenausgabe in Abbildung 5.13. Zusätzlich erfolgt vorab eine Konvertierung von möglichen Großbuchstaben in Kleinbuchstaben. Sowohl die einzelnen Klassen-, Attribut- als auch Methodennamen sind korrekt auslesbar. Auch Sonderzeichen (beispielsweise für Sichtbarkeiten) lassen sich fehlerfrei erkennen. Bei dem Namen des Interfaces ‚IAusleihe' hingegen ist ein Buchstabe falsch ausgelesen worden. Zur besseren Identifikation kann diese Diagrammkomponente vor ihrem eigentlichen Namen den Buchstaben ‚I' beinhalten. Wie aus dem exemplarischen UML-Klassendiagramm hervorgeht, ist dieser je nach Schriftart auch für das menschliche Auge nicht von einem klein geschriebenen ‚L' zu differenzieren.

Weiterhin ist es per Definition nicht möglich, ein Objekt eines Interfaces zu erzeugen. Die Angabe ‚abstract' wäre daher obsolet, obgleich sie durch Tesseract inklusive Sonderzeichen korrekt auslesbar ist. Bei Klassen hingegen ist diese Kennzeichnung essenziell. Alternativ würde die Modellierungssoftware abstrakte Klassen durch einen kursiv geschriebenen Klassennamen kennzeichnen. Dies ist mithilfe von Tesseract nicht erkennbar. Der Enterprise Architekt kann durch eine auswählbare Option alle als abstrakt gekennzeichneten Komponenten mit der Beschriftung ‚abstract' versehen.

Wie von Huber und Hagel (2020) bereits festgestellt, stellen sehr große Diagramme mit vielen Komponenten, kleiner Schrift und schlechter Bildqualität Herausforderungen für den Export von Textbausteinen dar. Auch sich stark überlappende Assoziationen, die auch aus menschlicher Sicht teils schwer nachvollziehbar sind, können fehlerhafte Zuordnungen zu Klassen mit sich bringen.

Das konzipierte Verfahren agiert bei der Objektdetektion zuverlässig, weist jedoch in wenigen Fällen Herausforderungen bei dem Textexport sowie der Zuordnung von Beziehungskomponenten auf. Für UML-Klassendiagramme mit wenigen Komponenten und gut lesbarer Schrift ist das Vorge-

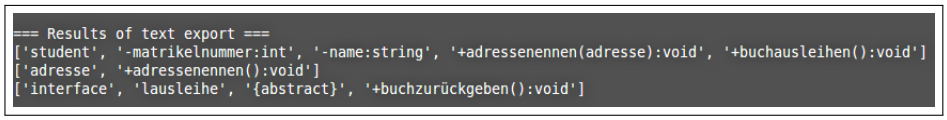


Abbildung 5.13: Exemplarische Extraktion von Textbausteinen aus UML-Klassendiagrammkomponenten für eine Evaluation der konzipierten Softwarekomponente.

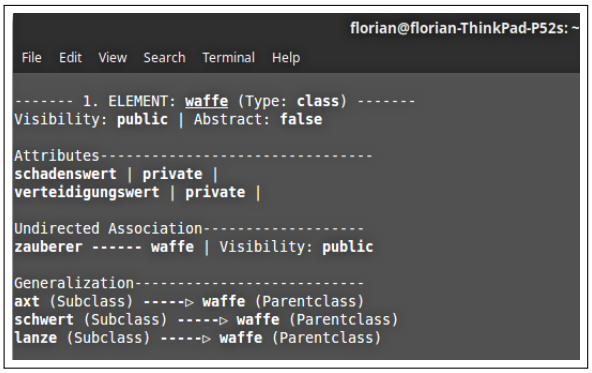


Abbildung 5.14: Auszug einer exemplarischen Konsolenausgabe für den Vergleich zweier unterschiedlicher UML-Klassendiagramme.

hen dagegen gut geeignet. Dennoch ist im Kontext der Arbeit die Verwendung von XML-Dateien für eine fehlerfreie Bereitstellung aller Informationen aus UML-Klassendiagrammen empfohlen.

Die Evaluation des Vergleichs von UML-Klassendiagrammen ist durch die Erstellung zweier XML-Dateien möglich. Erstere muss (ohne semantische Zusammenhänge zu beachten) alle im Kontext dieser Arbeit vergleichbaren Diagrammbestandteile beinhalten. Letztere kann beliebige Klassendiagrammkomponenten aufweisen, sofern diese nicht mit denen aus dem ersten Diagramm übereinstimmen.

Zunächst erfolgt die Übergabe des ersteren Diagramms als studentische Abgabe und gleichzeitig als Musterlösung. Die Softwarekomponente identifiziert in diesem Fall keinerlei Differenzen und zeigt somit, dass alle Diagrammbestandteile extrahiert, miteinander abgeglichen und keine Unterschiede erkennbar sind. Anschließend erfolgt ein Vergleich der vollkommen unterschiedlichen UML-Klassendiagramme. In diesem Fall schreibt der Prototyp alle identifizierten Diagrammbestandteile vollständig in die Konsole und kommuniziert so die Differenzen beider Diagramme zueinander. In diesem Schritt ist auch überprüfbar, ob alle Inhalte beider Diagramme er-

kannt und verglichen worden sind. Abbildung 5.14 zeigt exemplarisch einen Auszug aus einer solchen Konsolenausgabe. Zu sehen ist eine als fehlend identifizierte Klasse mit Namen ‚Waffe'. Innerhalb dieser befinden sich zwei private Attribute. Weiterhin weist sie eine ungerichtete Beziehung zu einer Klasse ‚Zauberer' auf und stellt die Elternklasse für die Subklassen ‚Axt', ‚Schwert' und ‚Lanze' dar.

Anhand der Konsolenausgabe ist überprüfbar, dass alle relevanten Diagrammbestandteile miteinander verglichen wurden und deren Differenzen identifizierbar sind. Gäbe es Komponenten, die nicht miteinander verglichen würden, so wären diese in der Ausgabe nicht enthalten. Da dies nicht der Fall ist, darf von einer korrekten Funktionsweise der Softwarekomponente ausgegangen werden.

5.4 Beantwortung von Forschungsfrage 1.2

In diesem Kapitel wurde eine Softwarekomponente für die Detektion und Lokalisation von UML-Klassendiagrammbestandteilen sowie für die Extraktion von Textbausteinen konzipiert und prototypisch implementiert. Dieser Schritt dient der Überführung relevanter Informationen in eine interne Repräsentation. Sofern eine XML-Datei und kein Bild als Eingabe vorliegt, sind Informationen ohne visuelle Untersuchungen exportierbar und in diese Struktur überführbar. Stehen zwei Diagramme in interner Repräsentation bereit, sind deren Komponenten miteinander abgleichbar sowie deren Differenzen kommunizierbar.

Das modular gehaltene Softwaredesign ist flexibel erweiterbar und lässt sich durch eine Instanziierung der Fassadeklasse mit weiteren Softwarebausteinen verknüpfen. Die Evaluation verdeutlicht, dass unter Verwendung von XML-Dateien alle kontextrelevanten UML-Klassendiagrammbestandteile identifizierbar und vergleichbar sind.

Die beschriebenen Ergebnisse stellen eine Antwort auf Forschungsfrage 1.2 dar und zeigen folglich auf, wie eine Software für den Vergleich studentischer UML-Klassendiagramme mit automatisch generierten Musterlösungen konzipiert und prototypisch implementiert werden kann.

KAPITEL 6

Feedback

In der hochschulischen Lehre von UML-Klassendiagrammen ist es Dozierenden kaum möglich, allen Studierenden ein vergleichbares Maß an individuellem und zeitnahem Feedback zu erstellten Lösungen bereitzustellen (vgl. Knobloch u. a., 2018). Aufgrund fehlender Erfahrung und Expertise ist es den meisten Studierenden zudem nicht möglich, ihre UML-Klassendiagramme selbst zu evaluieren und potenzielle Fehler zu identifizieren (vgl. Alhazmi u. a., 2021; vgl. Stikkolorum u. a., 2019). Während des Übungsbetriebs erhalten die Studierenden daher meist nur Rückmeldung des Typs ‚Richtige Antwort', selten jedoch vom Typ ‚Elaboriert' oder gar ‚Informatives Lehren' (vergleiche Tabelle 2.2). Sofern Dozierende elaboriertes Feedback ohne die Unterstützung einer Softwareanwendung zur Verfügung stellen, geschieht dies häufig zeitlich verzögert (vgl. Hasker und Rowe, 2011). Hattie und Timperley (2007) zufolge muss Feedback einen direkten zeitlichen Bezug zu einer vorausgegangenen Tätigkeit aufweisen, um effektiv zu sein. Der Mehrwert dieses individuellen, händisch angefertigten, elaborierten Feedbacks ist somit überschaubar. Dieses während des Übungsbetriebs zu gewährleisten ist kaum möglich (vgl. Huber und Hagel, 2022b).

Neben dem zeitlichen Faktor sind auch die transportierten Inhalte entscheidend. So ist beispielsweise Rückmeldung auf einem Selbst Level (siehe Abschnitt 2.4.4) vergleichbar effektiv wie ‚Kein Feedback' (siehe Tabelle

2.2). Auch die in Abschnitt 2.4.2 erörterten drei Fragen nach Hattie und Timperley (2007) spielen eine wichtige Rolle, wenn die Rückmeldung an die Studierenden effizient gestaltet werden soll. Dabei ist jedoch zu beachten, dass ein Übermaß an Informationen einen negativen Einfluss auf den Lernprozess nehmen kann (vgl. Jurgelaitis u. a., 2019). Die durch den Feedback Typ ‚Richtige Antwort' transportierten Informationen sind für viele Studierende ungenügend.

Das Adjektiv ‚effektiv' in Relation zu dem Substantiv ‚Feedback' ist im Folgenden an den Sprachgebrauch von Hattie und Timperley (2007) angelehnt. Da in dieser Arbeit keinerlei motivationspsychologische Untersuchungen stattfinden, sei darauf hingewiesen, dass besagtes Adjektiv nicht unter diesem Blickwinkel zu betrachten ist.

Automatisiertes, computergestütztes Feedback kann zu einer Individualisierung beitragen und die Inhalte dabei effektiv kommunizieren. Das Forschungsdesiderat der Frage 1.3 beinhaltet die Konzeption einer solchen Automatisierung. Diese soll in dem zu entwickelnden Gesamtkonzept Anwendung finden und somit Studierende in ihrem Lernprozess unterstützen sowie Dozierende entlasten. Letztere können aufgrund der gewonnenen Zeit Lehrinhalte vertiefen oder sich weiteren Themen widmen, die für die hochschulische Ausbildung von Software Engineering-Studierenden relevant sind.

6.1 Vorgehen und Einordnung in das Forschungsdesign

Auch dieser Abschnitt folgt dem von Hevner u. a. (2004) beschriebenen DSR-Forschungsdesign. Aus den zuvor angeführten realweltlichen Problemen können Anforderungen abgeleitet werden, die gemeinsam mit der in Kapitel 2 und 3 beschriebenen Wissensbasis als Entwicklungsgrundlage für das zu konzipierende Artefakt dienen. Wie in den vorherigen beiden Kapiteln werden auch hier die zu Forschungsfrage 1.3 in Relation stehenden Arbeiten angeführt (siehe Abschnitt 6.2), bevor eine Konzeption in Abschnitt 6.3 erfolgt. Das Artefakt verwendet die durch den Vergleich zweier UML-Klassendiagramme detektierten Differenzen und kommuniziert diese sowohl automatisiert als auch effektiv an Studierende und stellt gleichzeitig

die Ausgabe des Gesamtsystems dar. Eine Evaluation dieser Komponente ist daher nur in Kombination mit den in den vorherigen Abschnitten beschriebenen Systembestandteilen sinnvoll. Implementierung, Bewertung und Identifikation von Verbesserungspotenzialen befinden sich daher in Kapitel 7 und 8. Der letzte Abschnitt dieses Kapitels (6.6) ist der Beantwortung von Forschungsfrage 1.3 gewidmet.

6.2 Verwandte Arbeiten (Related Work)

Da Feedback in der Aus- und Weiterbildung von Studierenden eine entscheidende Rolle spielt, ist dieses Themengebiet nachvollziehbarerweise Gegenstand intensiver Forschung (vgl. Hattie und Timperley, 2007; vgl. Lipnevich und Panadero, 2021). Daher erfolgt in diesem Unterabschnitt eine Eingrenzung auf Feedback in der hochschulischen Lehre von Software Engineering und im speziellen auf computergestützte Rückmeldung für UML-Klassendiagramme. Auch motivationspsychologische Aspekte finden aufgrund der technischen Orientierung der Arbeit keine Berücksichtigung, könnten jedoch für potenzielle, zukünftige Weiterentwicklungen in den Fokus gerückt werden.

Eine von Dingsøyr (2021) verfasste Publikation beschreibt die agile Gestaltung eines Software Engineering-Kurses, in dem Studierende in iterativen Zyklen ihre erarbeiteten Ergebnisse präsentieren und dafür formative Bewertungen erhalten. Der Autor argumentiert, dass sich diese Form des regelmäßigen Feedbacks positiv auf den Lernerfolg von Studierenden auswirken kann (vgl. Dingsøyr, 2021). Auch ein von Müller-Amthor u. a. (2020) veröffentlichter Ansatz präsentiert einen agil gestalteten Software Engineering-Kurs. Die Ergebnisse der Publikation weisen ebenfalls darauf hin, dass bereitgestelltes Feedback Studierende positiv fördert (vgl. Müller-Amthor u. a., 2020). Neben den genannten Publikationen gibt es viele weitere, die sich mit nicht computergestütztem Feedback im Kontext von Software Engineering-Kursen befassen (vgl. Huber und Hagel, 2022b). Sie sind jedoch zu unspezifisch, um die Forschungsfrage 1.3 beantworten zu können und sind daher von den Inhalten der folgenden Konzeption abzugrenzen.

Weiterhin konnten Publikationen identifiziert werden, die im Rahmen von Software Engineering computergestütztes Feedback bereitstellen. So

entwickelten Farah u. a. (2022) einen Ansatz, der es Lehrpersonen durch die Unterstützung von Chatbots ermöglicht, Kursinhalte zu kommunizieren. Feedback für bearbeitete Übungsaufgaben zu erhalten ist damit jedoch nicht möglich (vgl. Farah u. a., 2022). Auch die in diesem Bereich publizierten Verfahrensweisen sind zu unspezifisch, als dass sie Forschungsfrage 1.3 hinreichend beantworten können.

Für die Bereitstellung von nicht computergestütztem Feedback in der hochschulischen Lehre von UML-Klassendiagrammen gibt es ebenfalls unterschiedliche Publikationen. So erläutern Rukmono und Chaudron (2022) beispielsweise einen Ansatz, bei dem Studierende einander Rückmeldung zu vorangegangenen Leistungen geben. Zusätzlich stellen die Autoren Richtlinien für die Anregung vom förderlichem Peer-Feedback bereit (vgl. Rukmono und Chaudron, 2022). Vergleichbare Ansätze sind durchaus digital abbildbar. Da sie es jedoch bis dato nicht sind, differenzieren sie sich von der zu konzipierenden Verfahrensweise.

Letztlich sind Softwareanwendungen zu nennen, die Studierenden eine orientierungsgebende Rückmeldung zu den von ihnen erstellten UML-Klassendiagrammen bereitstellen. Einige der in der SLR von Huber und Hagel (2022b) identifizierten und in Abschnitt 3.1 erörterten Publikationen befassen sich mit der Bereitstellung von Feedback für studentische UML-Klassendiagramme. Wie die vorherigen Kapitel implizieren, fokussiert sich die Arbeit auf eine individuelle Rückmeldung für Lernende anhand eines Vergleichs zweier UML-Klassendiagramme. Huber und Hagel (2022b) beschreiben 20 Softwarelösungen, die ein studentisches UML-Diagramm anhand eines vorgegebenen Regelwerks respektive einer Musterlösung bewertet und dazu Rückmeldung gibt. Tabelle 6.1 zeigt eine eigens erstellte Übersicht dieser Anwendungen, deren Typ des Feedbacks nach den Definitionen aus Tabelle 2.2 bestimmt wurde. Die Publikationen sind anhand der Autor*innen sowie der Jahreszahl identifizierbar und im Literaturverzeichnis inkludiert. Die Inhalte sind anhand des Feedback-Typs (nach der Hierarchie in Tabelle 2.2) sowie dem Erscheinungsjahr sortiert. Da computergestützte Rückmeldungen auf unterschiedliche Diagrammtypen übertragbar sind, enthält die Tabelle nicht ausschließlich Softwarelösungen, die auf UML-Klassendiagramme spezialisiert sein müssen (eine Betrachtung von UML-Diagrammen ist ausreichend).

Wie die Tabelle aufzeigt, bieten einige Anwendungen eine Verifizierung studentischer Diagramme in Form einer Note oder einer Punkteskala. Weitere bieten eine Fehlerkennzeichnung, bei der die Differenzen zu einem vorgegebenen Regelwerk oder einer Musterlösung kommuniziert werden. Einmal findet zusätzlich eine Verifizierung statt. Weitere Publikationen befassen sich mit einer Rückmeldung vom Typ ‚Elaboriert'. Hierbei erhalten Studierende eine Erklärung, warum ihre Antwort nicht korrekt ist. Lediglich zwei der Ansätze erreichen den komplexesten Typ ‚Informatives Lehren'. Für die zu erstellende Konzeption ist diese Art der Rückmeldung das Ziel. Sie bietet nicht nur elaboriertes Feedback sondern zusätzlich eine Verifikation, Fehlerkennzeichnung, strategische Hinweise sowie ggf. eine Musterlösung, die nach Definition in Tabelle 2.2 aber nicht erforderlich ist (vgl. Shute, 2008). Gerade diese beiden Publikationen sind daher explizit zu betrachten und von dem folgenden Vorhaben abzugrenzen.

Die von Hasker und Rowe (2011) publizierte Anwendung prüft studentische UML-Klassendiagramme anhand eines vordefinierten Regelwerks. Detektierte Abweichungen werden über eine Ausgabemaske anhand prägnanter Sätze kommuniziert. Ein weiterführender Link hinter den jeweiligen Anmerkungen führt zu einer detaillierten Beschreibung des Fehlers und zu Empfehlungen für die Verbesserung des Diagramms. Mit der Softwarelösung ist es nicht möglich zu verifizieren, ob Studierende die in einer textuellen Anforderungsbeschreibung enthaltenen Komponenten vollständig und korrekt in ein Klassendiagramm überführt haben und unterscheidet sich somit von der hiesigen Konzeptionierung.

Feedback vom Typ ‚Informatives Lehren' ist weiterhin in der Veröffentlichung von Bogdanova (2019) zu finden. Das Feedback-Modell folgt den von Hattie und Timperley (2007) vorgestellten drei Fragen für effektive Rückmeldung, die in Unterabschnitt 2.4.2 beleuchtet sind. Folgende Inhalte werden durch die Software unter Berücksichtigung dieser an die Studierenden kommuniziert (in Anlehnung an (Bogdanova, 2019, S. 3), übersetzt durch den Autor):

- Where am I going? - Die Studierenden erhalten einen Überblick der zu erreichenden Lernziele.
- How am I going? - Die Studierenden erhalten einen umfassenden Be-

richt über ihre Fehler.

- Where to next? - Die Studierenden erhalten anhand ihres Fehlerprofils eine Empfehlung, welche Kursinhalte und Aufgaben zu wiederholen sind.

Bogdanova (2019) bietet keine Rückmeldung in Form einer Verifizierung (beispielsweise durch eine Punkteskala) und keine Übersicht von Diagrammbestandteilen, die in einer textuellen Anforderungsbeschreibung zu extrahieren gewesen wären. Weiterhin ist keine (statistische) Übersicht studentischer Fehler für Lehrpersonen einsehbar. Anhand dieser Punkte sind die Softwarelösungen von den folgenden Inhalten der Arbeit zu differenzieren.

Publikation	**Feedback-Typ(en)**
Stikkolorum u. a., 2019	Verifizierung
Ichinohe u. a., 2019	Verifizierung
Boubekeur u. a., 2020	Verifizierung
Farias und Silva, 2020	Verifizierung
Soler u. a., 2010	Fehlerkennzeichnung
Cai u. a., 2011	Fehlerkennzeichnung
Hasker, 2011	Fehlerkennzeichnung
Herout und Brada, 2016	Fehlerkennzeichnung
Wei u. a., 2016	Fehlerkennzeichnung
Bian u. a., 2019	Fehlerkennzeichnung, Verifizierung
Modi u. a., 2021	Fehlerkennzeichnung
Schots u. a., 2010	Elaboriert
Sedrakyan und Snoeck, 2012	Elaboriert
Reischmann und Kuchen, 2016	Elaboriert
Ali u. a., 2018	Elaboriert
Saini u. a., 2019	Elaboriert
Huber und Hagel, 2020	Elaboriert
Alhazmi u. a., 2021	Elaboriert
Hasker und Rowe, 2011	Informatives Lehren

Tabelle 6.1: Klassifizierung von Feedback-Typen der von Huber und Hagel (2022b) identifizierten Softwareanwendungen, welche Vergleichsoperationen beinhalten.

Ziel der folgenden Konzeption ist das Aufgreifen der in Tabelle 6.1 zusammengefassten Ideen und Vorgehensweisen. Von besonderer Relevanz sind dabei die von Hasker und Rowe (2011) sowie Bogdanova (2019) präsentierten Vorgehensweisen bei der Bereitstellung von Feedback für Studierende. Sie werden im Folgenden aufgegriffen und erweitert.

6.3 Konzeption einer Feedback Richtlinie

Der von Huber und Hagel (2020) entwickelte Prototyp ist als erstes Artefakt im Sinne des zyklischen Entwicklungsverfahrens des DSR-Forschungsdesigns nach Hevner u. a. (2004) anzusehen. Das darin für Studierende bereitgestellte Feedback vom Typ ‚Elaboriert' kann sich positiv auf den Lernerfolg von Studierenden auswirken, bietet jedoch keine Verifikation und keine strategischen Hinweise für Verbesserung. Durch eine diesbezügliche Optimierung des Prototyps ist Feedback vom Typ ‚Informatives Lehren' generierbar. Die zu konzipierende Softwarekomponente führt selbst keine Vergleiche von UML-Klassendiagrammen durch. Es muss an dieser Stelle davon ausgegangen werden, dass Informationen und Resultate der zuvor entwickelten Softwarekomponenten vorliegen. Wie diese miteinander kombinierbar sind, ist in Kapitel 7 einsehbar.

Das von Hattie und Timperley (2007) präsentierte Feedback-Modell für die Verbesserung des Lernerfolgs (siehe Abbildung 2.14) soll als Grundlage herangezogen und auf die gegebene Fragestellung angepasst werden. Wie bei Bogdanova (2019) nachzulesen ist, eignet sich dieses Verfahrensmodell für den gegebenen Problemkontext. Abbildung 6.1 visualisiert das konzipierte Feedback-Modell. Als Zweck ist weiterhin die Reduzierung der Differenz zwischen dem aktuellem und dem gewünschten Wissensstand von Lernenden angeführt. Diese ist durch die Studierenden selbst, Dozierende oder das computergestützte Feedback realisierbar. Eine effektive Rückmeldung soll durch die Beantwortung der drei Fragen nach Hattie und Timperley (2007)

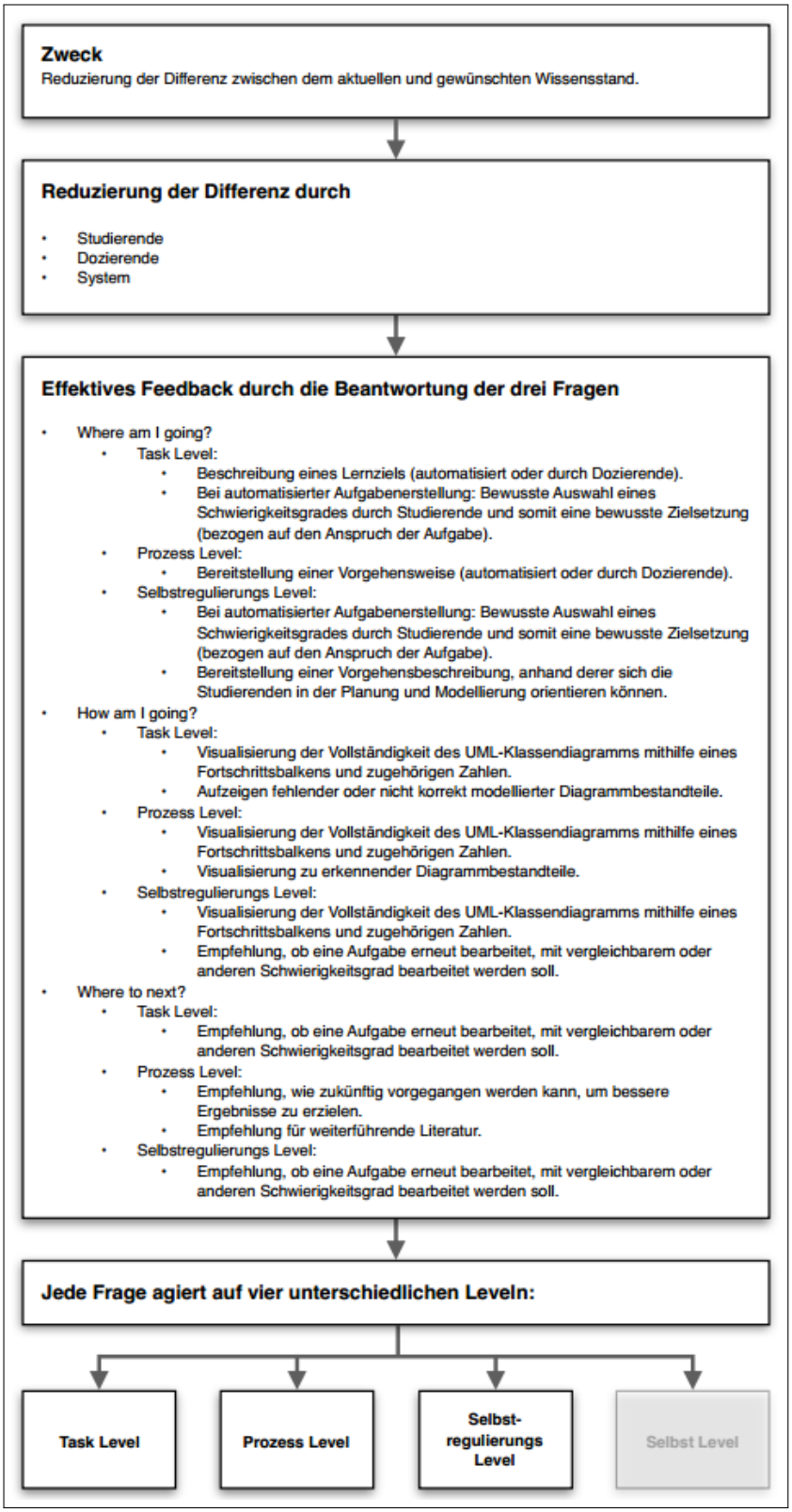


Abbildung 6.1: Eigenes Feedback Modell in Anlehnung an (Hattie und Timperley, 2007, S. 87).

auf einem Task, Prozess und Selbstregulierungs Level erfolgen. Auch die Anknüpfung des Selbst Levels ist theoretisch möglich, jedoch aufgrund des mangelnden inhaltlichen Bezugs für die Förderung des Lernprozesses von Studierenden wenig dienlich. Es findet daher keinen Einzug in diese Konzeption.

Innerhalb der ersten Frage wird im Rahmen des Task Levels eine Beschreibung des Lernziels kommuniziert. Dabei kann es sich um einen vorgefertigten oder einen von Lehrpersonen zu einer Übungsaufgabe bereitgestellten Text handeln. Es bestimmt, welches Ergebnis letztlich als erfolgreich zu verbuchen ist. Weiterhin können Studierende im Falle der automatisierten Generierung von Übungsaufgaben vorab einen Schwierigkeitsgrad auswählen, der ebenfalls Einfluss auf den Erfolg des Modellierungsprozesses nehmen kann und gleichzeitig auch als Selbstregulierung dient. Auf dem Prozess Level steht eine textuell vorhandene Vorgehensweise zur Verfügung, die eine strategische Planung des Erstellungsprozesses und gleichzeitig eine Selbstregulierung ermöglicht.

Durch die Maßnahmen für die Beantwortung der zweiten Frage erhalten Studierende auf einem Task Level eine Visualisierung der Vollständigkeit ihres Diagramms anhand eines Fortschrittsbalkens, der auf Basis errechneter Kennzahlen generierbar ist. Gleichzeitig wirkt diese auch auf dem Prozess sowie Selbstregulierungs Level. Zusätzlich können Studierende einsehen, welche individuellen Diagrammbestandteile nicht korrekt modelliert sind, fehlen oder überflüssig sind. Anhand dessen ist ableitbar, wie das vollständige und fehlerfreie Klassendiagramm aussieht. Den Studierenden wird jedoch bewusst keine Musterlösung in Form einer XML-Datei geboten. Vielmehr ist angedacht, dass sie ihre eigene Lösung anhand des Feedbacks erneut überarbeiten. Im Rahmen des Prozess Levels soll weiterhin eine Visualisierung aller zu extrahierender Diagrammbestandteile einsehbar sein. Daran können Studierende die Qualität ihrer eigenen Vorgehensweise ableiten und erkennen, wo ihnen ggf. ein Fehler unterlaufen ist. Innerhalb des Levels für die Selbstregulierung ist weiterhin eine Empfehlung zu nennen, die Studierenden mitteilt, ob zukünftig eine Übungsaufgabe mit niedrigerem, gleichem oder höherem Schwierigkeitsgrad bearbeitet werden sollte.

Die zuletzt genannte Information ist auch innerhalb des Task sowie Selbstregulierungs Levels der dritten Frage platzierbar. Auf einem Prozess

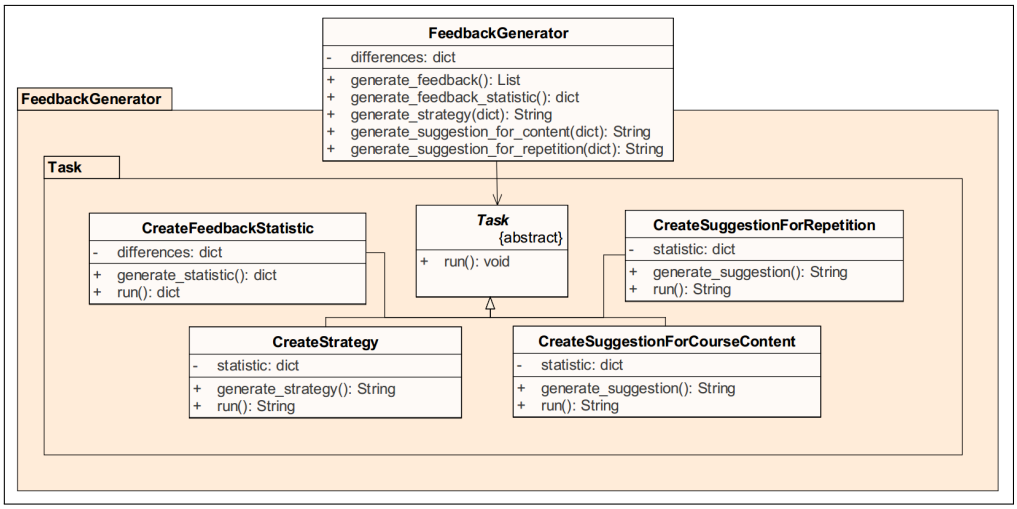


Abbildung 6.2: Softwaredesign der Softwarekomponente für die Bereitstellung von Feedback.

Level ist eine Empfehlung für das zukünftige Vorgehen sowie für weiterführende Literatur angedacht.

Wie Abbildung 6.1 aufzeigt, ist eine trennscharfe Einordnung der Bestandteile des Feedbacks in die jeweiligen Level nicht möglich, da sie gleichzeitig mehrere Aspekte bedienen. Nicht in der Darstellung enthalten sind Auswertungsmöglichkeiten für Lehrpersonen. Letztere sollen dabei eine Auflistung alle studentischer Fehler sowie eine Übersicht über die Anzahl der Diskrepanzen erhalten. Dafür ist eine Speicherung und Zurverfügungstellung einzelner Resultate notwendig.

Abbildung 6.2 visualisiert die Struktur der konzipierten Softwarekomponente in Form eines UML-Klassendiagramms. Es sind alle Klassen mit ihren wichtigsten Attributen und Methoden enthalten. Diese befinden sich in dem gemeinsamen Softwarepaket mit Namen ‚FeedbackGenerator' und sind über eine gleichnamige Fassadeklasse ansprechbar. Die Auswahl des entsprechenden Entwurfsmusters und die Kapselung unterschiedlicher Aufgabenbereiche in einzelne Klassen ermöglicht auch hier eine Entwicklung entsprechend der Prinzipien der Trennung von Zuständigkeiten (vgl. Vogel u. a., 2011) sowie der eindeutigen Verantwortung (vgl. Martin, 2018). Jede dieser Einheiten erbt von der abstrakten Klasse ‚Task' und befindet sich gemeinsam mit der Elternklasse in einem gleichnamigen Softwarepaket. An dieser Stelle ist davon auszugehen, dass die Differenzen zweier UML-Klassendiagramme bereits in Form einer Dictionary-Struktur vorliegen. Letztere

ist an anderer Stelle zu generieren und dient als Eingabeparameter für die Fassadeklasse ‚FeedbackGenerator'. Weiterhin erfolgt durch die jeweiligen Tasks keine (Konsolen-)Ausgabe, sondern die Rückgabe einzelner Bestandteile des Feedbacks in Form von Zeichenketten, die mit HTML-Annotationen versehen sind. Somit ist eine grafisch ansprechende Darstellung der produzierten Inhalte auf einer Weboberfläche realisierbar. Da zuvor detektierte Differenzen zwischen einem studentischen Diagramm und einer Musterlösung bereits zuvor in einer solchen Form speicherbar sind, ist in Abbildung 6.2 keine dedizierte Klasse hierfür enthalten.

Die Fassadeklasse besitzt eine Methode ‚generate_feedback()', welche die jeweiligen anderen in der Abbildung dargestellten Methoden aufruft. Diese instanziieren je einen abgeleiteten Task ($\widehat{=}$ Kindklasse) und geben als Rückgabewert einen Teil des gewünschten Feedbacks zurück. Sie stellen die folgenden Funktionalitäten bereit:

- ‚CreateFeedbackStatistic': Nimmt die detektierten Differenzen eines Vergleichsprozesses entgegen. Die Resultate stellen die Grundlage für die Generierung eines auf das individuelle Fehlerprofil von Studierenden angepasste Feedbacks dar. Die folgenden Kennzahlen werden erhoben und in Form einer Dictionary-Struktur von der ‚run()'-Methode zurückgegeben:
  - Anzahl der fehlenden Klassen in einem studentischen UML-Klassendiagramm.
  - Anzahl der nicht in einer Musterlösung vorhandenen Klassen, die innerhalb eines studentischen UML-Klassendiagramms modelliert sind.
  - Anzahl der Differenzen zweier Klassen mit demselben Klassennamen (es muss eine Klasse mit gleicher Namensgebung in beiden UML-Klassendiagrammen vorhanden sein).
  - Anzahl fehlender Attribute innerhalb des studentischen UML-Klassendiagramms.
  - Anzahl fehlender Methoden innerhalb des studentischen UML-Klassendiagramms.

  - Anzahl fehlender Beziehungen zwischen den jeweiligen Klassen innerhalb des studentischen UML-Klassendiagramms.
  - Gesamtanzahl der Fehler in einem studentischen UML-Klassendiagramm.

- ‚CreateStrategy': Nimmt die zuvor erstellte statistische Übersicht entgegen und erstellt Empfehlungen für das weitere Vorgehen anhand des individuellen Fehlerprofils der abgegebenen studentischen Lösung. Sofern die Gesamtzahl der Fehler unter drei liegt, meldet die Klasse nur einen allgemeinen Hinweis zurück, der besagt, dass die weiteren Inhalte des Feedbacks auf Verbesserungsmöglichkeiten evaluiert werden sollen. Andernfalls gibt es Hinweise, wie bei der Extraktion von Klassendiagrammkomponenten aus den textuellen Anforderungsbeschreibungen zukünftig optimaler verfahren werden kann. Weiterhin stellt die Klasse strategische Tipps für die korrekte Modellierung von Attributen, Methoden oder Beziehungen bereit, falls in einer dieser Kategorien mehr als drei Fehler detektiert werden konnten. Der Rückgabewert der ‚run()'-Methode ist eine Zeichenkette mit den individuellen Inhalten sowie HTML-Annotationen, um diese grafisch ansprechend auf einer Webseite einbetten zu können.

- ‚CreateSuggestionForCourseContent': Nimmt die zuvor erstellte statistische Übersicht entgegen und erstellt anhand der jeweiligen Fehlerhäufigkeit und -distribution eine Empfehlung für weiterführende Literatur. Diese ist durch den Autor vorselektiert, jedoch auf Codebasis beliebig erweiterbar. Zudem wird in jedem Fall darauf verwiesen, die Inhalte des Vorlesungsskripts zu repetieren. Liegt die Gesamtzahl der Fehler unter vier, besteht die Rückmeldung der Klasse lediglich aus allgemeinen, weiterführenden Literaturhinweisen in Form eines mit HTML-Annotationen versehenden Strings. Alternativ erfolgt eine Überprüfung, ob im Kontext von Klassen (fehlend oder überflüssig), Attributen, Methoden und Beziehungen mehr als drei Fehler detektiert wurden. Ist dies für eine oder mehrere der Kategorien der Fall, wird der String entsprechend erweitert und letztlich von der ‚run()'-Methode rückgemeldet.

- ‚CreateSuggestionForRepetition‘: Nimmt die zuvor erstellte statistische Übersicht entgegen und generiert anhand der Gesamtanzahl detektierter Fehler eine Empfehlung, ob zukünftig eine Aufgabe mit höherem (Kennzahl <= 3), vergleichbarem (Kennzahl > 3 & Kennzahl <= 10) oder niedrigerem (Kennzahl > 10) Schwierigkeitsgrad bearbeitet werden soll. Dieser Teil des Feedbacks stellt den Rückgabewert der ‚run()‘-Methode in Form eines Strings dar. An anderer Stelle ist zu prüfen, ob eine Erhöhung respektive Verringerung des Schwierigkeitsgrades möglich ist.

Die Fassadeklasse erstellt anhand der individuellen Rückmeldungen der Tasks eine Liste. Diese stellt in Kombination mit den detektierten Differenzen das Feedback dar. Zusätzlich sind diese Informationen anderweitig speicherbar und können als Grundlage statistischer Auswertungen studentischer Leistungen durch Dozierende herangezogen werden. Eine Generierung solcher ist an anderer Stelle vorzunehmen, da das vorgestellte Softwarepaket ausschließlich für die Rückmeldung an Lernende gedacht ist.

Aufgrund des modularen Aufbaus ist eine Erweiterung der bestehenden Funktionalität durch das Hinzufügen weiterer Klassen einfach realisierbar. Die Kommunikation zu anderen Softwarebausteinen ist über die Fassadeklasse möglich.

6.4 Implementierung

Codebeispiel 6.1 zeigt exemplarisch einen Auszug aus der Fassadeklasse ‚FeedbackGenerator‘. Deren Methode ‚generate_feedback()‘ ruft weitere Operationen auf, die jeweils einen der zuvor präsentierten Task instanzieren und daraufhin ‘run()‘ aufrufen. Beispielhaft ist dies für die Erstellung der Feedback-Strategie abgebildet. Dabei wird ein Objekt der Klasse ‚CreateStrategy‘ erzeugt und diesem die zuvor errechnete statistische Übersicht übergeben.

Besagte Klasse ist in Codebeispiel 6.1 exemplarisch enthalten. Die von der ‚run()‘-Methode aufgerufene Operation ‚generate_strategy(...)‘ führt die konzeptionell beschriebenen Abfragen durch und erstellt in Abhängigkeit des Fehlerprofils eine Rückmeldung (in Codebeispiel 6.1 enthalten ist eine if-Anweisung zur Prüfung der Anzahl identifizierter Fehler im Bezug auf

```
from FeedbackGenerator.Task.CreateFeedbackStatistic
                        import CreateFeedbackStatistic
from FeedbackGenerator.Task.CreateStrategy
                        import CreateStrategy
from FeedbackGenerator.Task.CreateSuggestionForCourseContent
                        import CreateSuggestionForCourseContent
from FeedbackGenerator.Task.CreateSuggestionForRepetition
                        import CreateSuggestionForRepetition

class FeedbackGenerator:

 def generate_feedback(self):

  statistic = self.generate_statistic()
  suggestions_for_strategy =
     self.generate_strategy(statistic)
  suggestion_to_repeat_course_contents =
     self.generate_suggestion(statistic)
  suggestion_to_repeat_or_get_new_exercise =
        self.generate_suggestion(statistic)
  ...
  return feedback_list


 def generate_strategy(self, statistic):

  generate_strategy = CreateStrategy(statistic)
  suggestions_for_strategy = generate_strategy.run()

  return suggestions_for_strategy
```

Codebeispiel 6.1: Beispielhafte Implementierung der Klasse ‚FeedbackGenerator‘.

```
from FeedbackGenerator.Task.Task import Task

class CreateStrategy(Task):

 def run(self):

  strategy = self.generate_strategy()

  return strategy


 def generate_strategy(self):

  strategy = ""
  ...
   ...
   if self.__statistic["nr_missing_attributes_over_all_classes"]
                                                          > 3:
    strategy +=
       "<u>Es wurden einige Fehler bei Attributen erkannt. ..."
  ...
  return strategy
```

Codebeispiel 6.2: Beispielhafte Implementierung der Klasse ‚CreateStrategy‘.

Attribute). Ein String mit HTML-Annotationen stellt den Rückgabewert dar.

Die Klasse ‚CreateFeedbackStatistic' beinhaltet lediglich eine Summation detektierter Differenzen und ist daher nicht explizit als Codebeispiel angeführt. Für die weiteren Klassen des Pakets (‚CreateSuggestionForCourseContent', ‚CreateSuggestionForRepetition') ist der Code vergleichbar mit dem in Exempel 6.1 und daher ebenfalls nicht zusätzlich angeführt.

6.5 Evaluation

Eine Evaluation der konzipierten Softwarekomponente mit Studierenden ist notwendig, um zu verifizieren, ob und wie effektiv das bereitgestellte Feedback sie in ihrem Lernprozess unterstützt. Jedoch ist eine Auswertung anhand der Umgebung im Sinne des Relevanz-Zyklus nach Hevner u. a. (2004) an dieser Stelle (noch) nicht durchführbar. Als Eingabewert müssen detektierte Differenzen zweier UML-Klassendiagramme vorhanden sein. Für die Ausgabe des Feedbacks ist eine Weboberfläche erforderlich, damit die annotierten Inhalte grafisch korrekt und ansprechend präsentierbar sind.

Eine Evaluation findet daher nach Zusammenführung der vorgestellten Softwarekomponenten sowie der Konzeption einer zugehörigen Weboberfläche statt und ist in Kapitel 8 ausführlich präsentiert.

6.6 Beantwortung von Forschungsfrage 1.3

In diesem Kapitel ist eine Konzeption und prototypische Implementierung für die Bereitstellung von Feedback des Typs ‚Informatives Lehren' präsentiert worden. Das von Hattie und Timperley (2007) veröffentliche und in Abbildung 2.14 gezeigte Modell zur Verbesserung des Lernerfolgs sowie in der Wissensbasis bestehende Ansätze lieferten die Grundlage für die Entwicklung des Artefakts. Es vereint die unterschiedlichen Charakteristiken dieser ineinander.

Das modulare Softwaredesign ermöglicht eine flexible Erweiterung bestehender Funktionalitäten. Es ist vorgesehen, dass eine Rückmeldung an Studierende über eine Weboberfläche erfolgt.

Die beschriebenen Resultate stellen die Antwort auf Forschungsfrage 1.3 dar und zeigt folglich auf, in welcher Form Studierenden automatisiert und computergestützt individuelles und effektives Feedback zu erstellten UML-Klassendiagrammen bereitgestellt werden kann.

KAPITEL 7

Konzeption und prototypische Implementierung

Nach der Planung und prototypischen Entwicklung einzelner Softwareartefakte in den vorangegangenen Kapiteln kann nun ein computergestütztes Gesamtsystem zur Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering konzipiert und exemplarisch implementiert werden. Besagte Artefakte sind darin entsprechend zu integrieren. Ihr Funktionsumfang ist jedoch noch nicht ausreichend, um die zentrale Forschungsfrage hinreichend zu beantworten. In diesem Kapitel sind daher umfassende Überlegung zur Bereitstellung dieser und weiterer Funktionalitäten mithilfe einer Benutzerschnittstelle sowie zugehörigem Backend angeführt.

7.1 Vorgehen und Einordnung in das Forschungsdesign

Das Vorgehen orientiert sich ebenfalls an dem DSR-Forschungsdesign nach Hevner u. a. (2004). Die Anforderungen und Wissensbasen aller vorherigen Kapitel dienen als Entwicklungsgrundlage und Ausgangspunkt für das Gesamtsystem. Für dieses erfolgt ebenfalls eine Konzeption (siehe Abschnitt 7.2) und prototypische Implementierung (siehe Abschnitt 7.4).

Erste Evaluationen zuvor entwickelter Systemkomponenten in der Um-

gebung sind bereits durchgeführt und die in Abbildung 1.2 beschriebenen Zyklen des DSR-Forschungsdesigns aktiv durchlaufen. Dies wiederholt sich, aufbauend auf den bis dato beschriebenen Ergebnissen in diesem Kapitel. Eine finale Beurteilung des Gesamtsystems findet sich in Kapitel 8.

7.2 Konzeption

Dieser Abschnitt fokussiert sich auf die Zusammenführung der vorgestellten Softwarekomponenten zu einer Backend-Struktur. Weiterhin ist eine interaktive Weboberfläche für die Bereitstellung von Inhalten und der Interaktion mit Studierenden oder Dozierenden zu entwickeln. Ziel dabei ist die Erfüllung der in Kapitel 3 erhobenen Anforderungen. Allerdings ist nicht jeder Funktionswunsch innerhalb dieser Arbeit berücksichtigt. Eine Stellungnahme zu nicht berücksichtigen Anforderungen findet sich ebenfalls in den folgenden Ausführungen.

Eine Unterteilung in Front- und Backend ist vorzunehmen, um zwei logisch voneinander separierbare Einheiten (Zurverfügungstellung der Kernfunktionalität sowie die Anzeige der Ergebnisse) auch physisch zu trennen. Die Funktionalitäten des Backends lassen sich auf unterschiedliche Weisen visualisieren und ggf. auch mit einem bereits existierenden Softwaresystem verknüpfen.

Obwohl die Anforderungen auch Wünsche von Lehrenden berücksichtigen, soll das Resultat dieses Abschnitts ein System für studierendenzentriertes Lehren und Lernen darstellen (siehe Abschnitt 2.3).

7.2.1 Nicht berücksichtigte Anforderungen

Die in Tabelle 7.1 aufgelisteten Anforderungen sind nicht in der folgenden Konzeption inkludiert. Aus Gründen der Nachvollziehbarkeit ist neben den Identifikationsnummern auch der Inhalt dieser angeführt. Der Wortlaut ist dabei identisch zu dem in Kapitel 3

Eine Visualisierung von Musterlösungen für Studierende nicht vorgesehen, da die Gesprächspartner*innen in den qualitativen, leitfadengestützten Experteninterviews (siehe Kapitel 3) explizit darauf hinwiesen, dass individuelles Feedback der simplen Ausgabe einer Musterlösung vorzuziehen ist

(siehe A8). Jedoch ist angedacht, dass Studierende detektierte Differenzen zwischen ihrer sowie der Musterlösung einsehen und somit den Aufbau letzterer nachvollziehen können.

Eine Rückmeldung an Studierende während des Modellierungsvorgangs ist unter Verwendung des Echtzeit-Objektdetektors YOLOv5 theoretisch möglich, erfordert jedoch ein eigenes Konzept und wird daher nicht berücksichtigt. Vielmehr sollen sich Lernende zuallererst ungestört selbst Gedanken über das zu erstellende Diagramm machen. Falls sie eine zwischenzeitliche Überprüfung wünschen, besteht die Möglichkeit für ein aktuelles Diagramm Feedback einzuholen und dann weiterzuarbeiten. So können sie ihren eigenen Lernprozess flexibel gestalten. Aus den genannten Gründen ist Anforderung A13 exkludiert.

Anhand der vorausgegangenen Konzeptionen einzelner Softwarekomponenten wird deutlich, auf welchen Aufgabentyp sich diese Arbeit fokussiert. Studierende erhalten eine Übungsaufgabe, modellieren die darin beschriebenen Diagrammbestandteile und können sich letztlich individuelle Rückmeldung einholen.

Weitere Aufgabentypen, die beispielsweise eine Verschriftlichung bestehender UML-Klassendiagramme erfordern (siehe A14), finden ebenfalls keinen Einzug in diese Konzeption. Auch die Anzeige respektive Umwandlung in Programmcode (oder vice versa) ist nicht vorgesehen (siehe A57, A58, A59, A60, A63, A69), da viele moderne Modellierungswerkzeuge diese Funktionalitäten bereits offerieren und sich die Studierenden in dieser Phase der Softwareentwicklung (siehe Abbildung 2.1) mit dem Design und nicht der Implementierung befassen sollen. Folglich sind auch spezielle Visualisierungen, wie beispielsweise 3D-Darstellungen nicht enthalten (siehe A62, A64, A65, A67, A68).

Bei dem Vergleich zweier UML-Klassendiagramme sind alle enthaltenen Diagrammkomponenten miteinander abzugleichen. Es ist jedoch fraglich, warum Letztere in einer textuellen Anforderungsbeschreibung enthalten sein sollen, jedoch keine Prüfung auf Vollständigkeit für diese erfolgt. Somit ist Anforderung A29 nicht berücksichtigt.

Da bestehende Modellierungsanwendungen bereits alle für den akademischen Kontext relevanten Funktionalitäten zur Erstellung von UML-Klassendiagrammen zur Verfügung stellen, ist keine Eigenentwicklung einer sol-

chen vorgesehen. Studentische Aktivitäten während der Modellierung in einer externen Softwareanwendung zu loggen (siehe A37), ist somit zum aktuellen Zeitpunkt nicht vorgesehen. Das System muss erstellte UML-Klassendiagramme auch nicht selbst in eine XML-Datei exportieren, da dies von den Softwareanwendungen bereits übernommen wird (siehe A38).

Eine Überprüfung von Synonymen beim Vergleich zweier UML-Klassendiagramme ist bewusst nicht in die Konzeption integriert (siehe A42). Studierende sollen sich an den Begrifflichkeiten orientieren, die innerhalb der textuellen Anforderungsbeschreibung verwendet wurden. Dies ermöglicht eine bewusste Schulung auf das genaue Lesen der Texte sowie das Verwenden vorgegebener Worte.

Ein Vergleich studentischer UML-Klassendiagramme anhand eines definierten Regelwerks ist überflüssig, sofern korrekte Musterlösungen vorhanden sind (A48). Fehler in studentischen Diagrammen lassen sich mithilfe dieser Vorgehensweise ebenso identifizieren.

Die grafische Darstellung zu erkennender Diagrammbestandteile innerhalb einer textuellen Anforderungsbeschreibung ist vorgesehen (in Abhängigkeit des gewählten Schwierigkeitsgrads). Die Studierenden sollen jedoch keinerlei Informationen darüber erhalten, in welchem Zusammenhang diese zueinander stehen (siehe A54). Ansonsten wäre ein UML-Klassendiagramm vollständig anhand dieser Informationen abbildbar und die Lernenden müssten keinerlei Eigenleistung mehr erbringen.

Alle in diesem Unterabschnitt nicht genannten Anforderungen sind durch die folgende Konzeption zu berücksichtigen. Es ist jedoch zu betonen, dass es sich bei dieser Arbeit um eine prototypische Implementierung eines Softwaresystems zur Unterstützung der hochschulischen Lehre von UML-Klassendiagrammen handelt. Die Zielsetzung fokussiert sich daher nicht auf die Bereitstellung eines vermarktbaren Produkts. Dafür ist das Hinzufügen weiterer Systembestandteile, wie beispielsweise ein Login mit Benutzer- und Rollenmanagement, notwendig. Das Softwaredesign muss diesbezügliche Erweiterungen ermöglichen. Jedoch erfolgt ausschließlich eine Umsetzung der in Kapitel 3 erhobenen, funktionalen Anforderungen.

ID	Inhalt	Exkludiert

A8	Das System muss Studierenden die Möglichkeit bieten, eine Musterlösung für eine angezeigte textuelle Aufgabe einzusehen.	✗
A13	Das System sollte Studierenden die Möglichkeit bieten, Feedback für ein UML-Klassendiagramm während des Modellierungsvorgangs zu erhalten.	✗
A14	Das System sollte Studierenden die Möglichkeit bieten, textuelle Aufgaben zu vorhandenen UML-Klassendiagrammen zu schreiben.	✗
A29	Das System sollte Dozierenden die Möglichkeit bieten zu spezifizieren, welche Komponenten eines UML-Klassendiagramms das System bei einem Vergleich berücksichtigen soll.	✗
A37	Das System sollte studentische Aktivitäten während der Modellierung eines UML-Klassendiagramms loggen.	✗
A38	Das System sollte die Möglichkeit besitzen, modellierte UML-Klassendiagramme in eine XML-Datei zu exportieren.	✗
A42	Das System sollte Synonyme für Klassen- , Attribut- und Methodennamen in der Bewertung berücksichtigen.	✗

A48	Das System sollte studentische UML-Klassendiagramme mit einem vordefinierten Regelwerk vergleichen und Abweichungen detektieren.	✗
A54	Das System sollte Studierenden die Möglichkeit bieten, zusammengehörige UML-Klassendiagrammkomponenten durch Markierung zu identifizieren.	✗
A57	Das System sollte Studierenden die Möglichkeit bieten, Programmcode für eine textuelle Anforderungsbeschreibung zu schreiben.	✗
A58	Das System sollte Studierenden die Möglichkeit bieten, Programmcode in transformierter Form als UML-Klassendiagramm einzusehen.	✗
A59	Das System sollte Studierenden die Möglichkeit bieten, automatisiert erzeugten Programmcode zu einem modellierten UML-Klassendiagramm einzusehen.	✗
A60	Das System sollte Studierenden die Möglichkeit bieten, den generierten Programmcode neben einem zugehörigen UML-Klassendiagramm anzuzeigen.	✗

A62	Das System sollte Studierenden die Möglichkeit bieten, ein 2D UML-Klassendiagramm in Augmented Reality Ansicht, 3D oder Virtual Reality Ansicht einzusehen.	✗
A63	Das System sollte Studierenden die Möglichkeit bieten, erstellten Programmcode auszuführen und die Veränderungen der Objektparameter zur Laufzeit anzuzeigen.	✗
A64	Das System sollte Studierenden die Möglichkeit bieten, mit erstellten UML-Klassendiagrammen in 2D, Augmented Reality, 3D oder Virtual Reality zu interagieren.	✗
A65	Falls Studierende ein 3D UML-Klassendiagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein 3D UML-Klassendiagramm zu konvertieren.	✗
A67	Falls Studierende ein Augmented Reality Diagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein Augmented Reality Diagramm zu konvertieren.	✗
A68	Falls Studierende ein Virtual Reality UML-Klassendiagramm wünschen, sollte das System ein 2D UML-Klassendiagramm in ein Virtual Reality UML-Klassendiagramm zu konvertieren.	✗

A69	Falls Studierende im Programmcode eine Zeile markieren, sollte das System zugehörige Diagrammbestandteile in dem zugehörigen UML-Klassendiagramm hervorheben.	✗

Tabelle 7.1: Innerhalb der Konzeption des Gesamtsystems nicht berücksichtigte Anforderungen.

7.2.2 Konzeption eines Backends

Das in Abbildung 7.1 dargestellte Softwaredesign zeigt den strukturellen Aufbau des Backends. Dessen Bestandteile sind, ganz im Sinne der Trennung von Zuständigkeiten (vgl. Vogel u. a., 2011) sowie Zuweisung eindeutiger Verantwortlichkeiten (vgl. Martin, 2018), in einzelne Softwarepakete untergliedert und in den folgenden Unterunterabschnitten näher erläutert. Wie gehabt, handelt es sich um keine vollständige Darstellung aller zu implementierender Diagrammbestandteile.

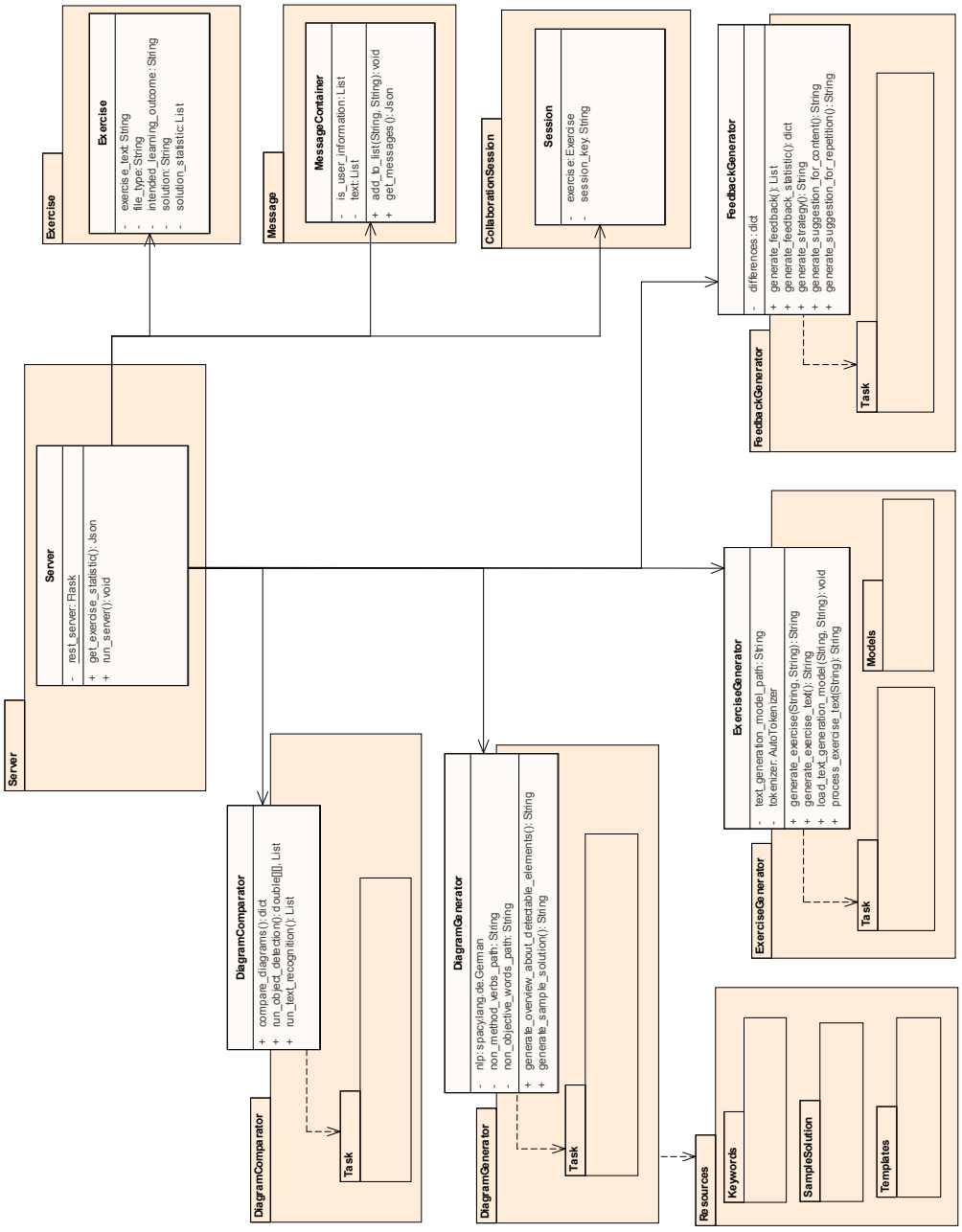


Abbildung 7.1: Softwaredesign für ein Backend.

7.2.2.1 Konzeption eines Servers

In Form einer Software sind Server dazu geeignet, Daten, Ressourcen oder weitere Funktionalitäten lokal oder innerhalb eines Netzwerks anzubieten. Anfragen von Clients lassen sich entgegennehmen, verarbeiten und beantworten. Ein solches Client-Server-Modell erlaubt die Trennung der eigentlichen Funktionalität und der grafischen Darstellung voneinander.

Wie in Kapitel 3 erläutert, fiel die Wahl der Programmiersprache auf Python. Auch die bereits vorgestellten Softwarekomponenten sind in darin entwickelt. Folglich soll die Programmiersprache auch für die Implementierung des Gesamtsystems herangezogen werden. Das für Python entwickelte Web Framework Flask[33] bietet anhand seiner unterschiedlichen Bibliotheken und Module eine simple Möglichkeit für die Entwicklung eines Servers, der über das Hypertext Transfer Protocol (HTTP) und dessen gängige Operationen, wie beispielsweise GET, PUT, POST oder DELETE, ansprechbar ist. Darüber lassen sich Informationen empfangen, versenden, aktualisieren und löschen (es gibt weitere ausführbare Aktionen, die jedoch für den gewünschten Kontext weniger Relevanz aufweisen). Es eignet sich daher optimal für die Umsetzung unterschiedlichster Projektvorhaben (vgl. Coburn, 2020; vgl. Modi u. a., 2022). Durch wenige Codezeilen kann der Server auf einem selbst definierten Port gestartet werden und ist fortan lokal ansprechbar. Auch ein Zugriff aus einem Netzwerk (beispielsweise einem Universitäts- oder Firmennetzwerk) ist möglich, sofern der Server sich auf einem Rechner innerhalb dieses Verbunds befindet und eine Kommunikation erlaubt ist. Die HTTP-Operationen sind mithilfe von Methoden abbildbar. Durch eine Annotation über dem Namen lassen sich diese mithilfe des Frameworks so modifizieren, dass sie HTTP-Anfragen empfangen und versenden können.

Ziel des Servers ist es, Anfragen aus dem Frontend entgegenzunehmen, zu verarbeiten und entsprechende Antworten respektive Daten zur Verfügung zu stellen. Somit gilt er als Kommunikationsschnittstelle zwischen der grafischen Benutzeroberfläche und den zuvor konzipierten Softwarekomponenten. Das in Abbildung 7.1 dargestellte Paket ‚Server' beinhaltet eine gleichnamige Klasse, die unter Zuhilfenahme des Flask-Frameworks HTTP-Anfragen aus einem lokalen Netzwerk ermöglicht. Gestartet wird er durch

[33] Dokumentation einsehbar unter: https://flask.palletsprojects.com/en/2.3.x/

einen Aufruf der Funktion ‚run_server()‘. Bei ‚get_exercise_statistic()‘ handelt es sich um eine von Flask modifizierte Methode die, sofern vorhanden, bereits errechnete, statistische Übersichten über die Differenzen studentischer Diagramme zu einer Musterlösung an ein Frontend kommunizieren kann. Als Rückgabetyp ist in Abbildung 7.1 ein Objekt im JavaScript Object Notation (JSON) Format angegeben. Fragt ein Client diese mithilfe einer GET-Operation an, bettet das Framework das Objekt in eine HTTP-Response ein und stellt diese zur Verfügung. Besagte Methode ist hier exemplarisch abgebildet. Da der gesamte Funktionsumfang über den Server kommuniziert werden muss, existiert eine Vielzahl weiterer, die jedoch alle nach einem vergleichbaren Prinzip agieren. Bereits vorhandene Informationen lassen sich durch das Frontend abrufen. Sind Erstellungs- oder Verarbeitungsschritte vonnöten, erstellen sie eine Instanz benötigter Klassen und nutzen deren Funktionsumfang, um die gewünschte Antwort zu liefern.

Die Objekte der Klassen ‚Exercise‘, ‚MessageContainer‘ sowie ‚Session‘ speichern relevante Informationen und stellen diese bei Bedarf bereit. Eine detaillierte Beschreibung findet sich in Unterunterabschnitt 7.2.2.3.

7.2.2.2 Integration konzipierter Softwarekomponenten

Die in den vorherigen Kapiteln konzipierten Softwarepakete ‚ExerciseGenerator‘, ‚DiagramGenerator‘, ‚DiagramComparator‘ sowie ‚FeedbackGenerator‘ finden sich in Abbildung 7.1 wieder. Aus Gründen der Übersichtlichkeit sind die bereits erläuterten Tasks darin nicht enthalten.

Lediglich das Paket ‚DiagramGenerator‘ wurde um eine Abhängigkeit zu dem Verzeichnis ‚Resources‘ erweitert. Innerhalb der darin angelegten Ordner befinden sich die folgenden Dateien:

- ‚Keywords‘: Enthält eine Ansammlung von Textdateien, die beispielsweise als Attribute zu klassifizierende, nichtgegenständliche Nomen hinterlegen.

- ‚SampleSolution‘: Beinhaltet eine generierte Musterlösung in Form einer XML-Datei. Diese ist abrufbar und kann über das Frontend verfügbar gemacht werden.

- ‚Templates‘: Umfasst eine XML-Datei, die als Grundlage für die Ge-

nerierung von Musterlösungen dient. Alle darzustellenden UML-Klassendiagrammkomponenten lassen sich in diese Schablone integrieren.

Die beschriebenen Ordner sind nicht neu hinzugekommen. Sie befanden sich zuvor lediglich in dem Paket ‚DiagramGenerator'. Durch eine Auslagerung sind diese theoretisch auch für weitere Programmbereiche verfügbar, ohne eine Fassadeklasse ansprechen zu müssen (deren Aufgabe nicht in der Verwaltung von Ressourcendateien liegt).

Untereinander weisen die Softwarepakete keine Abhängigkeiten auf. Der Gedanke eines modularen Aufbaus ist somit erhalten geblieben. Bei Bedarf sind die Funktionalitäten der Softwarepakete durch Erzeugung einer Instanz der jeweiligen Fassadeklasse zugänglich. Die Methoden der Klasse ‚Server' orientieren sich an diesem Vorgehen.

7.2.2.3 Weitere relevante Softwarepakete und Klassen

Abbildung 7.1 beinhaltet drei weitere Softwarepakete, die bis dato nicht erläutert sind. Sie dienen der Speicherung relevanter Informationen zur Laufzeit in Form von Objekten. Die erhobenen Anforderungen sehen beispielsweise vor, dass Dozierende ihren Studierenden eine selbst erzeugte Übungsaufgabe stellen können. Darin enthaltene Texte müssen durch den Server speicherbar und verfügbar sein. Durch die Attribute einer Klasse (die über ‚get()' und ‚set()' zugänglich sind) ist dies realisierbar.

In dem Paket ‚Exercise' befindet sich eine Klasse mit gleichem Namen, welche für die Speicherung aufgabenbezogener Informationen verantwortlich ist. Über die Attribute sind der Aufgabentext, eine zugehörige Musterlösung und deren Dateityp, das Lernziel sowie die statistische Auswertung (sofern bereits ein Vergleich stattgefunden hat) hinterlegt. Durch die Erzeugung eines Objektes bleiben die Inhalte zur Laufzeit verfügbar, sofern sie nicht durch eine neue Übungsaufgabe überschrieben werden.

Die Klasse ‚MessageContainer' ist in dem Paket ‚Message' hinterlegt und ist für die Sicherung von Textnachrichten in einem Chatverlauf dienlich. Als Attribute stehen Listen mit den einzelnen Nachrichten sowie der Information, ob diese von Studierenden oder Dozierenden verfasst worden ist, bereit. Innerhalb dieser Gruppen gibt es keinerlei Unterscheidung. Studierende können folglich anonym Nachrichten mit Fragestellungen oder Anmerkungen

verschicken.

Letztlich ist das Paket ‚CollaborationSession' mit seiner Klasse ‚Session' zu nennen. Die Anforderungen sehen eine Möglichkeit für eine kollaborative Zusammenarbeit von Studierenden vor. Innerhalb einer solchen Session ist eine Übungsaufgabe (in Form eines Objekts der Klasse ‚Exercise') sowie ein Identifikator gespeichert. Studierende können über Eingabe dieser Kennzahl einer Session beitreten und die entsprechende Aufgabe simultan bearbeiten. Eine gemeinsame Modellierung an einem UML-Klassendiagramm ist nicht vorgesehen, da die Studierenden explizit individuelles Feedback für ihre jeweilige Lösung erhalten sollen.

7.2.3 Konzeption eines Frontends

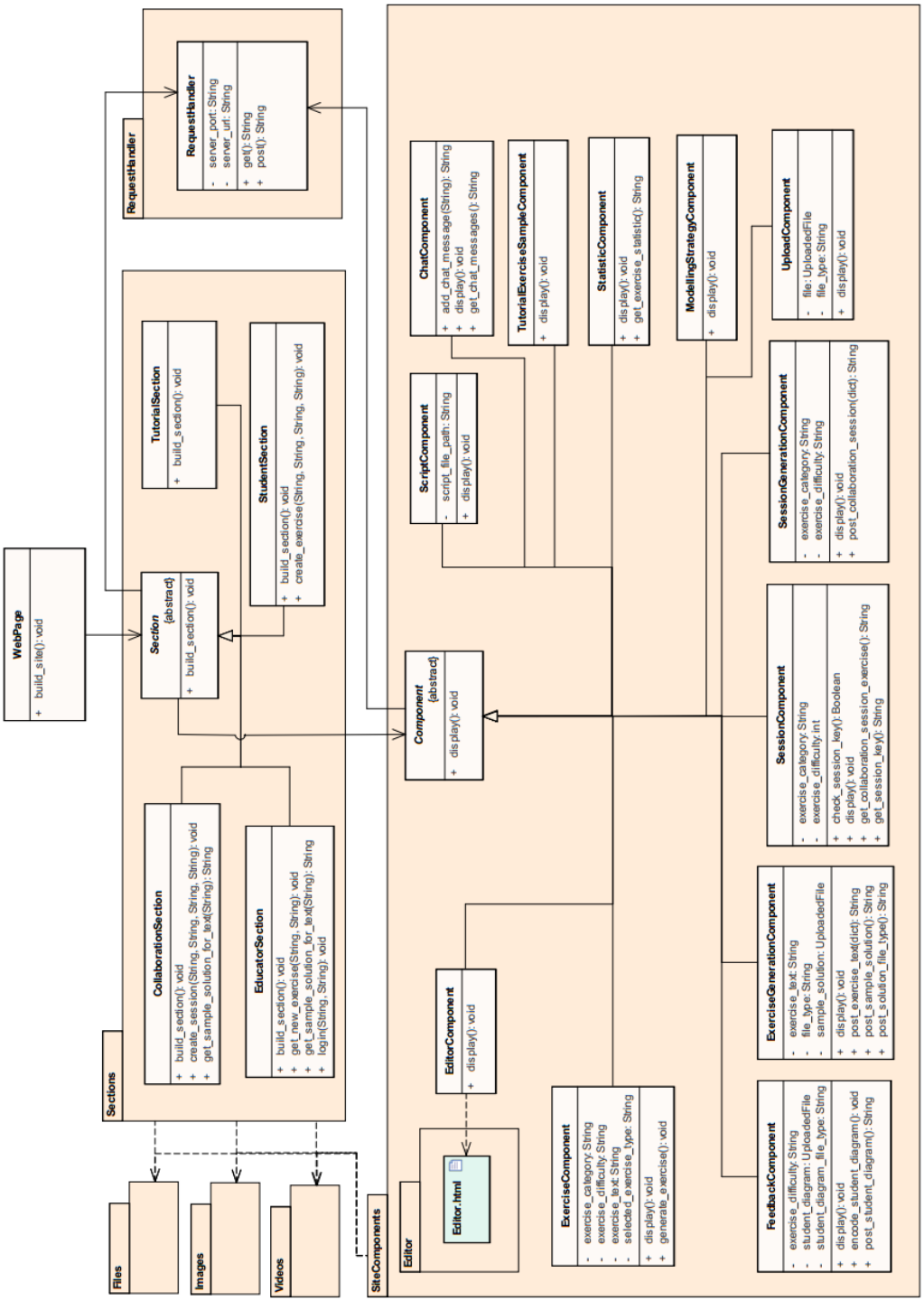


Abbildung 7.2: Softwaredesign für ein Frontend.

Das Softwaredesign des Frontends ist in Abbildung 7.2 dargestellt. Auch hier findet eine klare Trennung von Zuständigkeiten (vgl. Vogel u. a., 2011) sowie Zuweisung eindeutiger Verantwortlichkeiten (vgl. Martin, 2018) statt. Relevante Komponenten sind daher in unterschiedliche Pakete gegliedert, die durch die folgenden Unterunterabschnitte näher beleuchtet sind.

7.2.3.1 Konzeption einer Grundstruktur für eine interaktive Webseite

Die Entwicklung einer interaktiven und ansprechenden Webseite ist durch grundlegend unterschiedliche Herangehensweisen realisierbar. Gerade moderne Frameworks sind besonders geeignet, um diese schnell und oftmals ohne Interaktion mit HTML-, CSS- oder JavaScript-Dateien zu entwickeln. Die Python-Bibliothek Streamlit[34] ist ein repräsentativer Vertreter dieser Gruppe. Mithilfe von Python-Programmcode lassen sich Webseiten aufbauen, ohne viel Zeit in die Darstellungen und die hintergründigen Abläufe zu investieren. Der Fokus kann somit auf die Zurverfügungstellung der eigentlichen Inhalte gelenkt werden. Zu programmieren sind lediglich die Darstellungen und Bestandteile, die es zu visualisieren gilt. Alles weitere übernimmt das Framework selbstständig.

Die in Abbildung 7.2 dargestellte Klasse ‚WebPage' stellt den Einstiegspunkt für das Framework dar. Innerhalb der Methode ‚build_site()' sind die grundlegenden Inhalte der Webseite, die für alle Unterseiten simultan angezeigt werden, zu definieren. Da dieses Promotionsvorhaben in Kooperation der Hochschule für angewandte Wissenschaften Kempten sowie der Universität Regensburg entstand, finden sich die Logos dieser Einrichtungen im oberen Bereich aller Seiten wieder. Letztere sind über ein Navigationsmenü auf der linken Seite erreichbar. Nach einem Klick auf einen Menüeintrag wird innerhalb der Methode die Instanz einer Rubrik (im Klassendiagramm als ‚Sections' tituliert) erzeugt und anschließend ‚build_section()' aufgerufen. Weitere Details finden sich im folgenden Unterunterabschnitt.

Weiterhin steht auf der rechten Seite der Webseite ein Icon zur Verfügung, über das die Anwender*innen von Streamlit vorgegebene Konfigurationen (beispielsweise ein dunkles oder helles Seitenlayout) einstellbar sind.

[34] Einsehbar unter: https://streamlit.io/

7.2.3.2 Konzeption unterschiedlicher Rubriken

Aus den erhobenen Anforderungen sind vier Programmbereiche ableitbar, welche für die diversen Akteure verfügbar sein sollen. Diese befinden sich allesamt in dem in Abbildung 7.2 dargestellten Softwarepaket ‚Sections'. Darin befindet sich eine abstrakte Klasse mit Namen ‚Section', von der vier Kindklassen die Methode ‚build_section()' erben. Da in ‚WebPage' bereits das Grundgerüst für die angezeigte Webseite definiert wird, bauen die Sektionen unterschiedliche Rubriken mit diversen Inhalten auf, welche für die Erfüllung der Anforderungen notwendig sind. Die funktionsgebenden Bestandteile der Gesamtanwendung befinden sich, wie zuvor angeführt, im Backend. Dennoch soll an dieser Stelle eine tabellarische Auflistung der Inhalte einzelner Sektionen mitsamt einer Übersicht erfüllter Anforderungen erfolgen. Es ist explizit darauf hingewiesen, dass sich diese nur in Symbiose zwischen Front- und Backend erfüllen lassen. Lediglich ein konzipierter Login mit Benutzername und Passwort für Dozierende findet sich nicht in den Anforderungen wieder und wurde durch den Autoren hinzugefügt, um anzuzeigen, dass Studierende auf diesen Bereich keinen Zugriff haben sollten.

Inhalt	**Beschreibung**	**Anforderung(en)**
Login	Textfelder für die Eingabe von Benutzername (vordefiniert) und Passwort (vordefiniert).	-
Angestrebte Lernziele	Textfeld für die Eingabe der angestrebten Lernziele.	A21, A22, A23
Übungsaufgabe	Textfeld für die Eingabe einer textuellen Anforderungsbeschreibung sowie einer Aufgabenstellung.	A19, A20, A23
Erzeugung einer Übungsaufgabe	Auswahlfelder für ein Themengebiet und einen Schwierigkeitsgrad. Durch einen Klick auf einen Button kann automatisiert eine Übungsaufgabe generiert und in ein Textfeld geschrieben werden.	A23

Vorgehensweise	Textfeld für die Eingabe einer gewünschten Vorgehensweise respektive Strategie bei der Modellierung.	A23
Upload	Komponente für den Upload einer Musterlösung im XML- oder Bildformat.	A23, A24, A25, A39
UML-Editor	Komponente für die Anzeige eines open-source UML-Editors.	A23, A27, A28
Erzeugung einer Musterlösung	Automatisiertes Einlesen des Textfelds Übungsaufgabe und anschließende Generierung einer zugehörigen Musterlösung.	A23, A25, A26, A45
Chat	Chat-Fenster, über das Fragen von Studierenden beantwortet werden können.	A30, A31
Statistik	Anzeige von statistischen Auswertungen, nachdem Studierende ein oder mehrere UML-Klassendiagramme eingereicht haben.	A32, A33, A44

Tabelle 7.2: Konzipierung der Inhalte für die Rubrik ‚Dozierende‘.

Tabelle 7.2 beinhaltet die relevanten Inhalte der Rubrik für Dozierende. Nicht darin enthalten sind Bestandteile wie beispielsweise Überschriften oder Buttons. Da eine Integration einer Benutzerverwaltung nicht vorgesehen ist, sollen Felder für die Eingabe eines vordefinierten Benutzernamens sowie eines vordefinierten Passworts angezeigt werden. Erst nach korrekter Eingabe dieser lassen sich die weiteren in Tabelle 7.2 angeführten Inhalte einsehen. Zuerst ist ein Lernziel für Studierende bereitstellbar. Anschließend ist ein Textfeld für Übungsaufgaben visualisiert. Weiterhin ist nach Auswahl eines Themengebiets sowie eines Schwierigkeitsgrads die automatisierte Erstellung einer solchen möglich. Das Ergebnis wird in das bereits angeführte Textfeld eingefügt. Anschließend ist optional eine Beschreibung einer gewünschten Vorgehensweise hinzuzufügen. Weiterhin muss eine Musterlösung angeboten werden. Hierfür können Dozierende die Upload-Kom-

ponente verwenden. Ist noch keine Musterlösung für eine gegebene, textuelle Anforderungsbeschreibung vorhanden, kann diese mit einem open-source UML-Editor[35] oder automatisiert erzeugt werden.

Inhalt	**Beschreibung**	**Anforderung(en)**
Lernziel	Anführung eines allgemeinen Lernziels für die Rubrik.	A11, A49
Themengebiet	Auswahlfeld für die Wahl eines Themengebiets.	A3
Schwierigkeitsgrad	Auswahlfeld für die Wahl eines Schwierigkeitsgrads.	A2, A4
Erstellungsmethode	Auswahlfeld für die Wahl einer Erstellungsmethode.	A7
Session-Key	Textfeld für die Eingabe eines Identifikationsschlüssels für eine Lernsession.	A7
Angestrebte Lernziele	Textuelle Darstellung eines durch Dozierende vorgegebenen Lernziels (nur falls von Dozierenden spezifiziert).	A11, A49
Übungsaufgabe	Textuelle Darstellung einer Übungsaufgabe.	A10
Empfohlenes Vorgehen	Textuelle Darstellung eines durch Dozierende empfohlenen Vorgehens (nur falls von Dozierenden spezifiziert).	A49, A55
Zu erkennende Elemente	Falls der Schwierigkeitsgrad einfach gewählt wurde, ist eine grafische Darstellung der in einer textuellen Anforderungsbeschreibung enthaltenen Bestandteile einsehbar.	A36, A51, A66

[35]Beispielhaft wurde für die Konzeption die folgende Software gewählt: https://app.diagrams.net/

Strategie-hilfe	Textuelle Darstellung einer Vorgehensweise für die Extraktion von Diagrammkomponenten aus den textuellen Anforderungsbeschreibungen.	A49, A55
Skript	Einbettung eines Skripts in PDF-Format.	A34, A35, A49, A52, A53, A56
UML-Editor	Komponente für die Anzeige eines open-source UML-Editors.	A5, A6, A61
Upload	Komponente für den Upload eines studentischen UML-Klassendiagramms im XML- oder Bildformat.	A15, A16, A18, A39
Chat	Chat-Fenster, über das Fragen an Dozierende oder weitere Studierende gestellt werden können.	A12
Fortschritts-balken	Einordnung der studentischen Leistung anhand eines Fortschrittsbalkens.	A1, A40, A41, A44, A46
Fehlende Dia-gramm-komponen-ten	Anzeige der in einem studentischen UML-Klassendiagramm fehlenden Bestandteile.	A1, A17, A40, A44, A46, A47, A50
Überflüssige Dia-gramm-komponen-ten	Anzeige der nicht in einer Musterlösung, aber in einer studentischen Lösung enthaltenen Bestandteile.	A1, A17, A40, A44, A46, A47, A50
Differenzen gleicher Dia-gramm-komponen-ten	Anzeige von Differenzen zwischen zwei Klassen, mit demselben Klassennamen.	A17, A40, A44, A46, A47, A50

Zu erkennende Elemente	Darstellung der in einer textuellen Anforderungsbeschreibung enthaltenen Bestandteile.	A1, A40, A46
Vorgehensweise	Beschreibung von Verbesserungsmöglichkeiten bezogen auf die Vorgehensweise.	A1, A40, A46
Kursinhalte	Beschreibung von Kursinhalten, die Studierende anhand ihres individuellen Fehlerprofils wiederholen sollten. Auflistung weiterführender Literatur.	A1, A9, A40, A43, A46
Weiteres Vorgehen	Empfehlung für das weitere Vorgehen bezogen auf den Schwierigkeitsgrad der Aufgabenstellung.	A1, A40, A46

Tabelle 7.3: Konzipierung der Inhalte für die Rubrik ‚Studierende'.

Die Rubrik für Studierende stellt den Kern der Konzeption dar und ist mit ihren Inhalten in Tabelle 7.3 erläutert. Für Lernende ist zuallererst ein allgemeines Lernziel beschrieben. Darauf folgen Auswahlmöglichkeiten für das Themengebiet, den Schwierigkeitsgrad sowie die Erstellungsmethode. Letztere steuert, ob eine Übungsaufgabe mit zugehöriger Musterlösung automatisiert generiert oder lediglich geladen werden (vorausgesetzt, über die Rubrik für Dozierende respektive für die Kollaboration sind diese bereits hinterlegt). Ist die Erstellungsmethode für eine kollaborative Lernsession selektiert, erscheint zusätzlich ein Feld für die Eingabe eines eindeutigen Identifikationsschlüssels. Fällt die Wahl alternativ auf eine von Lehrpersonen definierte Übungsaufgabe, wird diese mitsamt einem angestrebten Lernziel (falls spezifiziert) sowie einem empfohlenen Vorgehen (falls spezifiziert) angezeigt. Andernfalls ist lediglich die textuelle Anforderungsbeschreibung mit Aufgabenstellung abgebildet. Sofern der einfachste Schwierigkeitsgrad durch die Studierenden gewählt wurde, können sie in einem ausklappbaren Feld die zu erkennenden Diagrammbestandteile der Aufgabe einsehen. Darunter befindet sich eine Strategiehilfe, die ein Vorgehen bei der Textanalyse sowie der Modellierung skizziert. Zusätzlich ist ein Skript in Form eines PDFs enthalten. Auch für die Studierenden steht ein UML-Editor

bereit. Nachdem das Klassendiagramm fertig gestellt ist, kann es über eine Upload-Komponente hochgeladen werden. Unter einem Chat-Fenster, in dem sie Fragen an Dozierende und andere Lernende stellen können, befindet sich ein Button, um Feedback einzuholen. Dessen erster Bestandteil ist ein Fortschrittsbalken, der anhand der Gesamtmenge der Fehler in vier Kategorien unterteilt und die Leistung mit entsprechenden Farben untermauert. Bei einer Gesamtfehlerzahl <= 3 ist sie grün hinterlegt und gibt die Überschrift „Super Leistung“ aus. Zwischen > 3 & <= 5 Fehlern ist sie ebenfalls grün, die Überschrift hingegen weist jedoch nur „Gute Leistung“ aus. Zwischen > 5 & <= 10 Fehlern ist sie orange mit der Überschrift „Mittelmäßige Leistung“. Alternativ ist der Fortschrittsbalken rot hinterlegt und gibt „Verbesserungswürdige Leistung“ an. Als weitere Rückmeldung sind fehlende bzw. überflüssig zu erkennende Diagrammkomponenten sowie eine Empfehlung von Verbesserungsmöglichkeiten (zu wiederholenden Kursinhalten und für das Weitere Vorgehen) gegeben.

Inhalt	**Beschreibung**	**Anforderung(en)**
Lehrvideo	Einbettung eines Lehrvideos zu UML-Klassendiagrammen.	A49, A52, A53, A56
Skript	Einbettung eines Skripts in PDF-Format.	A34, A35, A49, A52, A53, A56
Beispielaufgabe	Einfache Übungsaufgabe mit zugehöriger Beschreibung, wie die einzelnen Diagrammkomponenten identifizierbar sind und modelliert werden müssen.	A10, A49
Übungsaufgabe	Übungsaufgabe ohne zugehörige Musterlösung.	A10
UML-Editor	Komponente für die Anzeige eines open-source UML-Editors.	A5, A6, A61
Upload für Lösung	Komponente für den Upload einer verfügbaren Lösungsdatei im XML- oder Bildformat.	A15, A16, A39

Upload	Komponente für den Upload eines studentischen UML-Klassendiagramms im XML- oder Bildformat.	A15, A16, A18, A39
Fortschrittsbalken	Einordnung der studentischen Leistung anhand eines Fortschrittsbalkens.	A1, A40, A41, A44, A46
Fehlende Diagrammkomponenten	Anzeige der in einem studentischen UML-Klassendiagramm fehlenden Bestandteile.	A1, A17, A40, A44, A46, A47, A50
Überflüssige Diagrammkomponenten	Anzeige der nicht in einer Musterlösung, aber in einer studentischen Lösung enthaltenen Bestandteile.	A1, A17, A40, A44, A46, A47, A50
Differenzen gleicher Diagrammkomponenten	Anzeige von Differenzen zwischen zwei Klassen, mit demselben Klassennamen.	A17, A40, A44, A46, A47, A50
Zu erkennende Elemente	Darstellung der in einer textuellen Anforderungsbeschreibung enthaltenen Bestandteile.	A1, A40, A46
Vorgehensweise	Beschreibung von Verbesserungsmöglichkeiten bezogen auf die Vorgehensweise.	A1, A40, A46
Kursinhalte	Beschreibung von Kursinhalten, die Studierende anhand ihres individuellen Fehlerprofils wiederholen sollten. Auflistung weiterführender Literatur.	A1, A9, A40, A43, A46

Weiteres Vorgehen	Empfehlung für das weitere Vorgehen bezogen auf den Schwierigkeitsgrad der Aufgabenstellung.	A1, A40, A46

Tabelle 7.4: Konzipierung der Inhalte für die Rubrik ‚Tutorial‘.

Für die Abbildung von Kursinhalten sowie erste Übungsaufgaben (inklusive Musterlösung), ist Rubrik ‚Tutorial‘ vorhanden (siehe Tabelle 7.4). Darin befindet sich ein Lehrvideo zu UML-Klassendiagrammen (exemplarisch wurde eines von der Videoplattform YouTube gewählt[36]) sowie ein Skript. Auch die Einbindung von anderem, hochschulspezifischem Videomaterial wäre möglich. Es folgt eine simple Beispielaufgabe mit zugehöriger Musterlösung, deren Erstellungsschritte detailliert erläutert sind. Darunter befindet sich eine anspruchsvollere Übungsaufgabe mit einem UML-Editor. Es wird vorausgesetzt, dass den Studierenden für diese eine Musterlösung zur Verfügung steht, die sie mitsamt ihres eigenen Klassendiagramms hochladen können. Durch einen Klick auf einen Button generiert das System automatisiert Feedback in derselben Form, in der sie auch in der Rubrik für Studierende strukturiert ist.

Inhalt	**Beschreibung**	**Anforderung(en)**
Übungsaufgabe	Textfeld für die Eingabe einer textuellen Anforderungsbeschreibung sowie einer Aufgabenstellung.	A2, A4, A7
Erzeugung einer Übungsaufgabe	Auswahlfelder für ein Themengebiet und einen Schwierigkeitsgrad. Durch einen Klick auf einen Button kann automatisiert eine Übungsaufgabe generiert und in ein Textfeld geschrieben werden.	A2, A3, A4, A7
Upload für Lösung	Komponente für den Upload einer verfügbaren Lösungsdatei im XML- oder Bildformat.	A7, A15, A16, A39

[36]Einsehbar unter: https://www.youtube.com/watch?v=ws_KI4Wun4I

Erzeugung einer Musterlösung	Automatisiertes Einlesen des Textfelds Übungsaufgabe und anschließende Generierung einer zugehörigen Musterlösung.	A7, A45
Session-Key	Textfeld für die Eingabe eines Identifikators für die zu erstellende Session.	A7

Tabelle 7.5: Konzipierung der Inhalte für die Rubrik ‚Kollaboration'.

Die Rubrik ‚Kollaboration' dient der Generierung einer Lernsession, um die Zusammenarbeit von Studierenden an einer Aufgabe zu ermöglichen. Tabelle 7.5 fasst die Inhalte zusammen. Übungsaufgaben lassen sich manuell eingeben oder (nach Auswahl eines Themenschwerpunkts sowie eines Schwierigkeitsgrads) automatisiert erzeugen. Musterlösungen sind über eine Upload-Funktion hochzuladen oder maschinell zu generieren. Letztlich muss die Angabe eines Identifikationsschlüssels erfolgen, um eine Lernsession eindeutig zu identifizieren. Dieser ist in der Rubrik ‚Studierende' anzugeben, damit Aufgabenstellung und Musterlösung der spezifischen Session geladen werden können.

Sich nicht wiederholende Darstellungen, wie beispielsweise die visualisierten Überschriften der Rubriken, übernehmen die Klassen aus dem Softwarepaket ‚Sections'. Es gibt jedoch viele in den vorangegangenen Tabellen erläuterte Komponenten, die in unterschiedlichen Programmbereichen auftreten. Sie jedes Mal neu zu codieren, verstößt gegen das Don't Repeat Yourself (DRY) Prinzip, welches besagt, dass sich identische Codebausteine nicht an unterschiedlichen Stellen im Programm befinden sollten (vgl. Thomas und Hunt, 2019; vgl. Martin, 2018). Kommt es zu einer Änderung, müssten alle diese Codebestandteile einzeln überarbeitet werden. Aus diesem Grund wurde das Softwarepaket ‚SiteComonents' konzipiert, dessen Beschreibung im folgenden Unterunterabschnitt enthalten ist.

7.2.3.3 Konzeption unterschiedlicher Plug-and-Play Komponenten

Innerhalb des genannten Softwarepakets befindet sich eine Klasse ‚Component' von der die einzelnen Komponenten die Methode ‚display()' erben. Jede Komponente erhält eine einzige sowie individuelle Aufgabe (und entsprechen somit den bereits mehrfach genannten Softwarearchitekturprinzpien). Für die Anzeige der Komponenten in einer Rubrik muss lediglich an der gewünschten Stelle die Instanz einer der Kindklassen erstellt und deren ‚display()'-Methode aufgerufen werden (die Einbindung ist also nach dem Plug-and-Play Prinzip möglich). Besteht Änderungsbedarf gibt es genau eine Klasse, an der die Anpassungen vorzunehmen sind. Nicht jede Komponente muss zwangsläufig eine grafische Darstellung bereitstellen. In diesen Fällen konsolidieren sie Informationen und unterstützten bei der Übertragung dieser an das Backend.

Im Folgenden ist eine Auflistung der in Abbildung 7.2 spezifizierten Komponenten einsehbar. Da zwölf Stück anzuführen sind, soll an dieser Stelle nur eine kurze Erläuterung der Aufgabe der Klasse stattfinden:

- ‚ExerciseComponent': Anzeige einer Übungsaufgabe. Wurde diese von Dozierenden erzeugt, ist zusätzlich ein angestrebtes Lernziel sowie ein empfohlenes Vorgehen abgebildet (sofern definiert).

- ‚ExerciseGenerationComponent': Stellt keine grafische Darstellung bereit. Die Klasse konsolidiert relevante Informationen bezüglich einer Übungsaufgabe und unterstützt bei der Weiterleitung an das Backend.

- ‚FeedbackComponent': Darstellung der (konzeptionell) entwickelten Feedback-Struktur.

- ‚SessionComponent': Anzeige einer in der Rubrik ‚Kollaboration' erstellten Übungsaufgabe.

- ‚SessionGenerationComponent': Stellt keine grafische Darstellung bereit. Die Klasse konsolidiert relevante Informationen bezüglich einer anzulegenden Session und unterstützt bei der Weiterleitung an das Backend.

- ‚UploadComponent‘: Stellt eine Möglichkeit für das Hochladen von UML-Klassendiagrammen bereit. Diese können entweder durch das Ziehen einer Datei mit der Maus auf ein angezeigtes Feld oder über einen Dateiexplorer hinzugefügt werden.

- ‚ModellingStrategyComponent‘: Stellt ein mögliches strategische Vorgehen bei der Erstellung von UML-Klassendiagrammen mithilfe von Texten und Bildern dar.

- ‚StatisticComponent‘: Anhand der gespeicherten Fehlerstatistiken erzeugt die Klasse eine grafische Darstellung dieser (für Dozierende). Darin enthalten ist zum einen eine Auflistung aller studentischen Fehler. Zum anderen ist ein Balkendiagramm abgebildet, welche die Anzahl der Fehler pro verglichenem Diagramm in vier Kategorien darstellt:

  - Anzahl Fehler $<= 3$.
  - Anzahl Fehler > 3 & $<= 5$.
  - Anzahl Fehler > 5 & $<= 10$.
  - Anzahl Fehler > 10.

- ‚TutorialExerciseSampleComponent‘: Anzeige einer simplen Übungsaufgabe mitsamt einem vorgegebenen Lösungsweg und der Abbildung des gewünschten UML-Klassendiagramms.

- ‚ChatComponent‘: Bereitstellung einer Nachrichtenfunktion, über die sich alle an einer Session beteiligten Akteure miteinander austauschen können.

- ‚ScriptComponent‘: Anzeige eines Vorlesungsskripts in PDF-Format. Wie in einem PDF-Viewer lassen sich alle Seiten anzeigen, drucken etc.

- ‚EditorComponent‘: Einbindung und grafische Darstellung eines opensource Modellierungseditors. Diese Klasse besitzt eine Abhängigkeit zu einer HTML-Datei in einem Unterpaket mit Namen ‚Editor‘. Letzteres betten den Editor in eine HTML-Datei ein, die schließlich von der Klasse ‚EditorComponent‘ dargestellt wird.

7.2.4 Kommunikation zwischen Backend und Frontend

Der Begriff Representational State Transfer (REST) umfasst eine Ansammlung von Designkriterien, anhand derer sich die Architektur einer Software ausrichten kann (vgl. Richardson und Ruby, 2007). Entwickelt wurden diese für eine effektive Verwaltung von Kommunikation in komplexen Netzwerken (wie beispielsweise dem Internet). Client und Server sind hierbei voneinander getrennt und können, im Falle von Webapplikationen, über das HTTP-Protokoll miteinander kommunizieren (vgl. Richardson und Ruby, 2007). Eines der relevantesten Kriterien hierbei ist die sogenannte Zustandslosigkeit dieser Entitäten (vgl. Richardson und Ruby, 2007). Ein Nachrichtenaustausch findet über definierte Schnittstellen statt und ist vollkommen unabhängig von vorherigen Anfragen.

Die konzipierte Architektur richtet sich nach diesen Kriterien und stellt somit eine RESTful (vgl. Richardson und Ruby, 2007) Webanwendung dar. Die im Backend verankerte Klasse ‚Server' definiert mithilfe der Flask-Bibliothek Methoden, die durch Angabe der Server-URL, dem Server-Port sowie dem individuellen Methodenpfad unter Verwendung des HTTP-Protokolls abrufbar sind. Für die Erstellung und Verarbeitung solcher Anfragen steht im Frontend das Softwarepaket ‚RequestHandler' mit einer gleichnamigen Klasse bereit. Sowohl die Sektionen als auch die Komponenten können auf diese zugreifen, benötigte Parameter bereitstellen und die Antworten des Servers entsprechend verarbeiten respektive anzeigen. Ein konkret nachvollziehbares Beispiel für eine mögliche Implementierung dieses Konzepts findet sich in Unterabschnitt 7.4.5. Alle Daten und Inhalte sind als JSON-Objekte codiert.

7.3 Designentscheidungen für die grafische Benutzerschnittstelle

Anhand der erhobenen Anforderungen ließen sich bis dato vier Rubriken sowie deren jeweilige Inhalte ableiten. Dies geschah aus einer funktionsgebenden und technischen Perspektive. Bevor jedoch eine prototypische Umsetzung des konzipierten Frontends erfolgen kann, sind wichtige Designentscheidungen im Hinblick auf die grafische Benutzerschnittstelle zu treffen.

Diese stellt den zentralen und alleinigen Berührungspunkt der Studierenden und Lehrenden mit der geplanten Softwarelösung dar. Sie muss dabei optisch ansprechend, vor allem jedoch leicht verständlich und intuitiv navigierbar sein, damit sie ohne nennenswerte Einstiegshürden verwendbar ist. Dies ist gerade im Hinblick auf Studierende sehr wichtig. Ihr Fokus sollte auf der Modellierung von UML-Klassendiagrammen anhand von Übungsaufgaben liegen und nicht auf dem Erlernen der Funktionsweise des Prototyps. Letzterer soll sie in ihrem Lernprozess unterstützen und darf folglich keine Einstiegshürde darstellen.

Das konzipierte Softwaredesign ermöglicht eine flexible Komposition von Komponenten innerhalb der vier Rubriken. Diese sind mitsamt ihren bereitzustellenden Inhalten von den in Kapitel 3 erhobenen Anforderungen abgeleitet. Die jeweiligen Komponenten lassen sich aufgrund des Softwaredesigns zwar beliebig anordnen, jedoch ist eine definierte Sortierung notwendig. Sie sollen die Anwender*innen je nach Rolle bei der Erzeugung von Übungsaufgaben, der Aneignung von Wissen zu UML-Klassendiagrammen, der Erstellung von kollaborativen Aufgaben oder der Abstraktion und Modellierung anhand textueller Anforderungsbeschreibungen unterstützen. Die Reihenfolge der in den Rubriken darzustellenden Komponenten kann daher nicht beliebig erfolgen. Für Studierende ist beispielsweise zuallererst eine Übungsaufgabe anzuzeigen. Anschließend sind Hilfestellungen sowie Möglichkeiten zur Modellierung und Abgabe von UML-Klassendiagrammen einzubinden. Eine Umkehrung dieser Anordnung ist aufgrund des zugrundeliegenden Lehrkontexts nicht sinnvoll.

Dennoch ist es von zentraler Bedeutung, die Benutzerschnittstelle so zu implementieren, dass sie von allen Hauptakteuren ohne umfassende Schulungen nutzbar ist. Die in diesem Abschnitt genannten grafischen Designentscheidungen für das Frontend orientieren sich an den Richtlinien und Regeln von Shneiderman u. a. (2017). Die Autoren beschreiben, dass in einem ersten Schritt eine Untersuchung der Zielgruppe sowie deren Fertigkeiten in Bezug auf die Nutzung von Benutzerschnittstellen zu tätigen ist. Die Zielgruppen sowie deren individuellen Anforderungen sind durch die vorangegangenen Kapitel bereits umfassend untersucht. Da es sich bei allen Personen um Studierende aus informatiknahen Studiengängen respektive um Dozierende solcher Studiengänge handelt, kann eine technische Affini-

tät vorausgesetzt werden. Erklärungen (beispielsweise durch Pop-up-Fenster) zur Funktionsweise eines Navigationsmenüs, von Auswahlfeldern oder Buttons ist folglich nicht notwendig. Eine klar verständliche Beschriftung aller Inhalte sowie eine intuitive Menüführung sind hingegen unabdingbar (vgl. Shneiderman u. a., 2017). Die Anwender*innen sind anhand logischer und für diesen Lehrkontext üblicher Schritte (beispielsweise die Bereitstellung sowie Bearbeitung einer Übungsaufgabe und letztlich das Feedback zu selbiger) durch die einzelnen Rubriken zu führen. Laut Shneiderman u. a. (2017) erzeugt eine effektive Benutzerschnittstelle ein positives Gefühl von Kompetenz und führt dazu, dass selbige gerne verwendet wird. Daher sind Designentscheidungen an dieser Stelle unabdingbar.

Die Zurverfügungstellung des Frontends soll in Form einer Weboberfläche erfolgen. Gegenüber einer Desktopanwendung sind hier keine Installationen notwendig. Weiterhin sind sie unabhängig von Betriebssystemen und weiteren Softwareinstallationen zugänglich. Lediglich ein Webbrowser ist notwendig. Diese Entscheidung macht den Prototyp respektive dessen Benutzerschnittstelle für die Hauptakteure leicht zugänglich.

Über ein Navigationsmenü sollen die folgenden vier Rubriken aufrufbar sein:

- Dozierende.
- Studierende.
- Tutorial.
- Kollaboration.

Für Anwender*innen muss es jederzeit zur Verfügung stehen. Dennoch darf es nicht zu viel Raum einnehmen, damit der Hauptfokus bei den eigentlichen Inhalten der Benutzerschnittstelle bleibt. Daher ist eine Navigationsleiste auf der linken Seite vorgesehen. Durch einen Klick soll diese ein- und ausgeklappt werden können. Ein ausgewählter Menüpunkt ist farblich zu hinterlegen. So ist bei der Verwendung des Prototyps nachvollziehbar, welche Rubrik gerade angezeigt wird. Farblich soll sich die Navigationsleiste durch einen hellen Grauton von der weißen Hintergrundfarbe der Rubriken

abheben. Dies ermöglicht einen guten Kontrast zu den dargestellten Inhalten und verbessert so beispielsweise die Leserlichkeit der Texte. Diese Bestandteile sind unabhängig der Benutzerinteraktionen immer identisch. Je nach Auswahl des Menüpunktes gilt es jedoch, die Inhalte der Rubriken in der korrekten Reihenfolge anzuzeigen. Die entsprechenden Informationen hierfür sind den Tabellen 7.2, 7.3, 7.4 und 7.5 zu entnehmen. Da diese Arbeit in Kooperation der Universität Regensburg (Deutschland) mit der Hochschule für angewandte Wissenschaften Kempten (Deutschland) entstand, sollen die Logos dieser Institutionen in der obersten Zeile aller Rubriken abgebildet werden.

Sowohl die Bereiche für das Tutorial als auch für die Kollaboration sind für Lernende vorgesehen. Gerade im Hinblick auf den definierten Problemkontext ist jedoch angedacht, dass sie überwiegend die Rubrik mit der Bezeichnung ‚Studierende' verwenden. Sie ist von zentraler Bedeutung für Lernende und daher mit diesem Namen versehen. Die Namensgebung ist nicht nur für das Navigationsmenü vorgesehen, sondern wird auch in den folgenden Kapiteln der Arbeit verwendet.

Die Python-Bibliothek Streamlit stellt vordefinierte Bausteine, wie beispielsweise Eingabe- oder Auswahlfelder, bereit. Über eine Konfiguration ist die Farbgebung all dieser Bausteine adaptierbar (eine farbliche Anpassung einzelner Bestandteile ist durch die Bibliothek nicht vorgesehen). Weitere visuelle Manipulationen nur mit dem Hinzufügen von zusätzlichen HTML- und CSS-Codes möglich. Sofern es sich nicht um Text handelt, kann jeder Baustein mit einem Text respektive einer Kurzbeschreibung versehen und somit von anderen unterschieden werden.

Shneiderman u. a. (2017) nennen acht Regeln, die es bei der Erstellung einer grafischen Benutzeroberfläche zu beachten gilt. Für die praktische Realisierung des Prototyps spielen diese eine zentrale Rolle. Die folgende Auflistung beinhaltet sowohl eine kurze Erklärung als auch eine prägnante Erläuterung, wie diese Regeln konkret Anwendung finden soll (in Anlehnung an (Shneiderman u. a., 2017, S. 96f.), übersetzt durch den Autor):

1. Konsistenz anstreben:
   Die Webanwendung soll identische Handlungsabläufe und Bestandteile identisch abbilden. Dies gilt auch für Farben, Layouts, Schriftgrößen etc.

Für gleichartige Inhalte ist eine durchgängige Darstellung geplant. Mit einer Unveränderlichkeit des grundlegenden Aufbaus der Seite (inklusive des Navigationsmenüs) geht Konsistenz einher. Für eine Abgrenzung und Erläuterung der Inhalte sind Titel, Subtitel und normaler Text einzusetzen. Die Streamlit-Bibliothek zeigt gleichartige Bausteine (beispielsweise Auswahlfelder) immer gleich an.

2. Personen unterscheiden sich:
   Unterschiedliche Benutzergruppen weisen diverse Anforderungen auf, die nicht identisch oder ähnlich sein müssen. Dies gilt auch innerhalb dieser Gruppierungen. Diese Anforderungen und Differenzen sind durch die Webanwendung zu beachten.
   Durch die vorangegangene Anforderungserhebung sind viele Anforderungen der unterschiedlichen Hauptakteure durch den Prototyp berücksichtigt. Da eine Anwendung in informatiknahen Studiengängen vorgesehen ist, darf von einer technischen Affinität ausgegangen werden.

3. Interaktionen geben informative Rückmeldungen:
   Die Webanwendung soll den Anwender*innen Rückmeldung auf vorangegangene Interaktionen zur Verfügung stellen.
   Der Prototyp stellt für alle ausführbaren Aktionen direktes Feedback bereit. Alle dynamischen Bausteine, wie beispielsweise Auswahlfelder, stellen eine sofortige Reaktion auf Interaktionen bereit. Auch bei der Ausführung fehlerhafter Aktionen muss der Prototyp Hinweise in Form von Fehlermeldungen bieten. So ist es Studierenden beispielsweise nicht möglich Feedback zu erhalten, solange sie kein UML-Klassendiagramm hochgeladen haben.

4. Dialoge so gestalten, dass sie zu einem Abschluss führen:
   Die Webanwendung soll innerhalb ihrer Dialoge klare Start- und Endpunkte definieren. Für Anwender*innen muss verständlich sein, in welchem Prozessschritt sie sich zu einem bestimmten Zeitpunkt befinden und wann dieser abgeschlossen sind.
   Die Inhalte der Rubriken orientieren sich in ihrer Reihenfolge an den von der Umgebung (im Sinne des DSR-Forschungsdesigns nach Hevner

u. a. (2004)) spezifizierten Schritten. So erhalten beispielsweise Studierende zu Beginn einer Übungseinheit eine textuelle Anforderungsbeschreibung, die es zu bearbeiten gilt. Nach der Modellierung erhalten sie Feedback. Der Prototyp muss Lehrende und Lernende schrittweise durch ihre jeweiligen Prozesse führen und verdeutlichen, in welchem Prozessschritt sie sich gerade befinden. Häufig ist diesen Personengruppen doch bereits aus Erfahrungswissen klar, welche Schritte als nächstes folgen könnten und wann der Prozess abgeschlossen ist (bei Studierenden beispielsweise nach dem Erhalt des Feedbacks).

5. Fehler vermeiden:
   Die Webanwendung soll so konzipiert und implementiert sein, dass Anwender*innen keine Fehler erzeugen können. Auch Programmabstürze und Exceptions sind zu vermeiden.
   Zur Einhaltung dieser Regel sind fehlerhafte Benutzereingaben durch den Prototyp abzufangen und mit einer informativen Rückmeldung zu beantworten.

6. Aktionen sollen einfach rückgängig gemacht werden können:
   Es kann vorkommen, dass Anwender*innen eine Aktion fälschlicherweise durchführen und diese rückgängig machen möchten. Die Webanwendung soll dies berücksichtigen.
   Der Prototyp soll nach der inkorrekten Durchführung einer Aktion zu jeder Zeit die erneute Ausführung und somit eine Überarbeitung vorheriger Interaktionen ermöglichen. So müssen beispielsweise von Studierenden hochgeladene UML-Klassendiagramme beliebig austauschbar sein, falls versehentlich eine falsche Datei ausgewählt wurde.

7. Internen Kontrollfluss unterstützen:
   Die Webanwendung soll komplexe und überraschende Darstellungen sowie Abläufe vermeiden. Die Anwender*innen müssen wichtige Informationen auf den ersten Blick erkennen können.
   Durch einen klar strukturierten Aufbau der Rubriken sind alle Prozessschritte chronologisch angeordnet. Mithilfe eindeutiger Beschriftungen soll der Prototyp schrittweise durch diese führen und folglich einen nachvollziehbaren Kontrollfluss erzeugen.

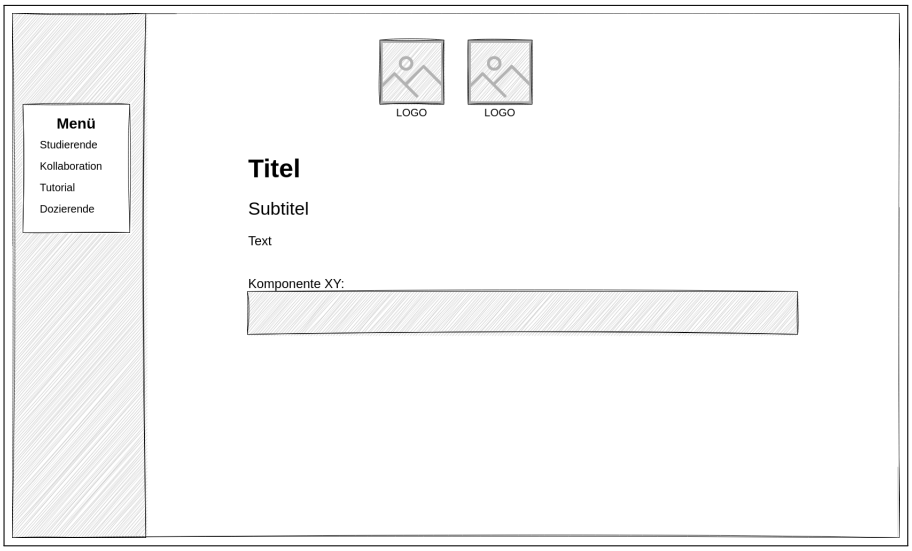


Abbildung 7.3: Skizzierung der grafischen Darstellung der Benutzerschnittstelle.

8. Belastung des Kurzzeitgedächtnisses verringern:
   Anwender*innen können sich häufig nur eine beschränkte Anzahl an Informationen merken. Die Webanwendung soll benötigte Informationen daher gut sichtbar darstellen.
   Der Prototyp soll alle benötigten Inhalte und Texte für alle Prozessschritte zur Verfügung stellen.

Obwohl die Benutzerschnittstelle explizit auf die Verwendung mittels eines Computers und zugehörigen Monitors ausgelegt ist, erlaubt die Streamlit-Bibliothek ohne zusätzlichen Code auch eine Nutzung auf Tablets und Smartphones. Die Darstellungen sind gegenüber der Bildschirmgröße adaptiv. Diese Funktionalität soll nach Implementierung des Prototyps jedoch nicht explizit getestet werden.

Abbildung 7.3 zeigt eine elektronisch erstellte Skizze des bereitzustellenden Frontends. Die zuvor genannten Punkte sind darin enthalten. Die Navigationsleiste mit ihren vier Menüpunkten ist auf der linken Seite einsehbar. Die Inhalte (inklusive der Logos der Institutionen) befinden sich mittig. Unter Beachtung der angeführten Richtlinien und Regeln soll die konzipierte Benutzerschnittstelle nun prototypisch implementiert werden.

7.4 Prototypische Implementierung

Dieser Abschnitt beinhaltet mehrere Codebeispiele, welche die Implementierung des konzipierten Systems exemplarisch verdeutlichen. Die Unterabschnitte sind dabei in Back- und Frontend unterteilt.

7.4.1 Implementierung des Backends

Die Veranschaulichung der Codiermöglichkeiten für das Backend ist in drei Unterunterabschnitte gegliedert und demonstriert die Implementierung des Flask-Servers mit einer beispielhaften Methode, die Klasse ‚Exercise' sowie den Start der resultierenden Software über eine Konsole. Weitere in Abbildung 7.1 dargestellte Softwarepakete sind durch die vorherigen Kapitel bereits ausführlich erläutert und finden daher keine erneute Berücksichtigung.

7.4.1.1 Implementierung des Servers

Der in Beispiel 7.1 abgebildete Programmcode stellt einen Auszug aus der Klasse ‚Server' dar. Darin befindet sich das Klassenattribut ‚rest_server', welches eine Instanz der Klasse Flask darstellt. Während der Erstellung eines neuen Objekts des Servers ruft der Konstruktor (‚__init__()') die Methode ‚run_server()' auf. Mithilfe des Klassenattributs kann durch die Angabe eines beliebigen, freien Ports der Flask-Server gestartet werden. Weiterhin kommt es zum Einsatz, um Python-Operationen zu annotieren und sie somit über den Server von außen ansprechbar zu machen. Wie das Codebeispiel verdeutlicht ist dafür die Angabe eines eindeutigen URL-Pfades, sowie einer HTTP-Operation notwendig. Enthalten ist exemplarisch eine POST- und eine GET-Operation, die jeweils den Text einer von Dozierenden erstellten Übungsaufgabe entgegennehmen und speichern respektive diesen auf Anfrage zurückgeben. Die Speicherung beziehungsweise der Abruf dieser Information gelingt unter Zuhilfenahme einer Instanz der Klasse ‚Exercise' (hierzu im folgenden Unterunterabschnitt mehr).

```
from flask import Flask, request, Response, jsonify
...
from Exercise.Exercise import Exercise
from CollaborationSession.Session import Session
from Message.MessageContainer import MessageContainer
from ExerciseGenerator.ExerciseGenerator import ExerciseGenerator
from FeedbackGenerator.FeedbackGenerator import FeedbackGenerator
from DiagramGenerator.DiagramGenerator import DiagramGenerator
...

rest_server = Flask("SystemBackend")

class Server:

 def __init__(self):

  self.run_server()

 def run_server(self):

  rest_server.run(port=4711)


 @rest_server.route('/post_exercise_text_from_educator',
                                              methods=['POST'])
 def post_exercise_text_from_educator():

  ...
  exercise.set_exercise_text(data["exercise_text"])
  ...

  return Response(response=response_string, status=200,
                              mimetype="application/json")


 @rest_server.route('/get_exercise_text_from_educator',
                                              methods=['GET'])
 def get_exercise_text_from_educator():

  ...
  response_string = exercise.get_exercise_text()

  return Response(response=response_string, status=200,
                              mimetype="application/json")
```

Codebeispiel 7.1: Beispielhafte Implementierung der Klasse ‚Server‘.

```
class Exercise:

 def __init__(self):

  self.__exercise_text = ""
  ...


 def set_exercise_text(self, text):

  self.__exercise_text = text


 def get_exercise_text(self):

  return self.__exercise_text
```

Codebeispiel 7.2: Beispielhafte Implementierung der Klasse ‚Exercise‘.

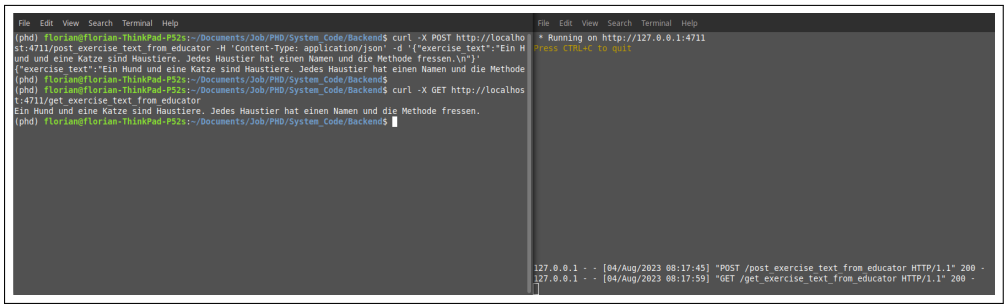


Abbildung 7.4: Start des konzipierten Servers auf einem Rechner.

7.4.1.2 Implementierung weiterer relevanter Softwarepakete und Klassen

Die in dem gleichnamigen Softwarepaket enthaltene Klasse ‚Exercise' ist in Codebeispiel 7.2 exemplarisch abgebildet. Sie besteht aus verschiedenen privaten Attributen (hier dargestellt ‚__exercise_text'), die über ‚get()'- sowie ‚set()'-Methoden abrufbar respektive veränderbar sind. Ein vergleichbarer Aufbau findet sich auch bei ‚MessageContainer' und ‚Session'. Daher sind diese nicht durch ein zusätzliches Codebeispiel demonstriert. Objekte der drei Klassen speichern relevante Informationen zur Laufzeit. Selbige werden durch den Server hinterlegt und bei Bedarf abgerufen.

7.4.1.3 Start des Servers auf einem Rechner

Da der konzipierte Server auf Basis der Python-Programmiersprache implementiert ist, kann er auf allen gängigen Betriebssystemen gestartet werden, auf denen entsprechende Installationen der benötigten Bibliotheken vorhanden sind. Eine zentralisierte Zurverfügungstellung auf hochschulinterner Serverhardware ist folglich problemlos realisierbar. Für Demonstrationszwecke wird im Kontext der Arbeit ein Rechner mit einem Linux Mint 19 Betriebssystem herangezogen. Nach Navigation in das Verzeichnis ‚Server' mit der Konsole ist ein Start über den folgenden Befehl möglich:

```
python3 Server.py
```

Der Server ist nun über die URL *http://127.0.0.1:4711* erreichbar und kann mit lokal installierten Applikationen interagieren. Da im Sinne der REST-Kriterien eine strikte Trennung von Client und Server besteht, lässt sich die Funktionsweise von letzterem durch einfache HTTP-Requests von

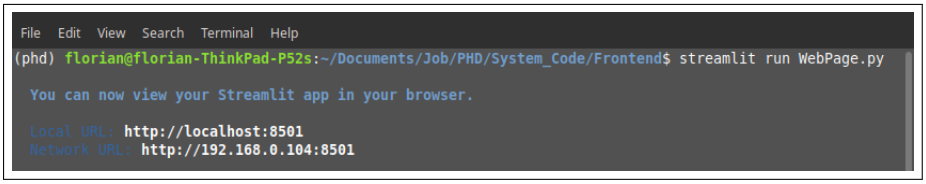


Abbildung 7.5: Konsolenbefehl und -ausgabe für den Start des Frontends.

der Konsole aus überprüfen. Mittels einer POST-Operation ist die in Codebeispiel 7.1 angeführte Methode ‚post_exercise_text_from_educator()' ansprechbar. Eine Speicherung des im JSON-Format empfangenen Texts verläuft wie beschrieben. Anschließend ist die Zeichenkette über eine GET-Operation abrufbar. Abbildung 7.4 beinhaltet die entsprechenden Konsolenbefehle und verdeutlicht, dass der Server die gewünschte Funktionsweise bereitstellt.

7.4.2 Implementierung des Frontends

Die Erläuterung für eine Implementierung des Frontends ist in vier Unterunterabschnitte gegliedert. Zuallererst erfolgt die Entwicklung einer Grundstruktur für die interaktive Webseite. Darauf aufbauend sind die jeweiligen Rubriken mit ihren Komponenten adaptierbar. Letztlich ist die Kommunikation zwischen Backend und Frontend aufgezeigt.

7.4.2.1 Implementierung einer Grundstruktur für eine interaktive Webseite

Wie Codebeispiel 7.3 aufzeigt, ist das Grundgerüst einer Webseite dank der Streamlit-Bibliothek vergleichbar einfach umsetzbar. Die abgebildete Methode ‚build_site()' baut in einem ersten Schritt das Navigationsmenü auf. Dieses befindet sich auf der linken Seite und beinhaltet die vier Menüpunkte: ‚Studierende', ‚Kollaboration', ‚Tutorial' und ‚Dozierende'. Die Seite registriert einen Mausklick auf eine der Auswahlmöglichkeiten, erstellt eine entsprechende Instanz einer Rubrik und ruft schließlich deren ‚build_section()'-Methode auf (anhand von Codebeispiel mit der Sektion für Studierende demonstriert). Weiterhin verdeutlicht der Programmcode, wie die Logos der genannten Institutionen inkludierbar sind.

Unter der Voraussetzung, dass die Klasse innerhalb einer Main-Methode

```
import streamlit as st
from streamlit_option_menu import option_menu

from Sections.StudentSection import StudentSection
from Sections.EducatorSection import EducatorSection
from Sections.CollaborationSection import CollaborationSection
from Sections.TutorialSection import TutorialSection

class WebPage:

 def build_site(self):

  with st.sidebar:
   selected = option_menu(
    menu_title="Menü",
    options=["Studierende", "Kollaboration",
             "Tutorial", "Dozierende"],
    icons=["person-circle", "person-plus-fill",
           "mortarboard-fill", "shield-lock"]
   )

  col1, col2, col3, col4 = st.columns([1, 1, 1, 1])
  col2.image("Images/LogoHS.png", use_column_width=True)
  col3.image("Images/ur.png", use_column_width=True)

  if selected == "Studierende":
        student_section = StudentSection()
        student_section.build_section()

  ...
```

Codebeispiel 7.3: Beispielhafte Implementierung der Klasse ‚WebPage‘.

instanziiert und ‚build_site()‘ aufgerufen wird, ist das Frontend über den in Abbildung 7.5 enthaltenen Konsolenbefehl zu starten. Lokal ist es nun unter der URL *http://127.0.0.1:8501* in einem Webbrowser aufrufbar. Abbildung 7.6 zeigt auf, wie das Grundgerüst der Benutzerschnittstelle ohne die Einbindung von Rubriken dargestellt wird.

7.4.3 Implementierung unterschiedlicher Rubriken

Die Rubrik für Studierende ist durch die Klasse ‚StudentSection‘ umgesetzt und in Codebeispiel 7.4 in kondensierter Form dargestellt. Als Spezialisierung der ‚Section‘ erbt sie die Methode ‚build_section‘. Mithilfe der Streamlit-Bibliothek visualisiert sie die Überschrift „Software Engineering Übung“, beschreibt ein Lernziel und fügt anschließend die benötigten Komponenten ein, indem sie entsprechende Objekte erzeugt und ‚display()‘ aufruft. Die Erstellung individueller Texte, Buttons etc. wird direkt von der jeweiligen Sektion übernommen. Sich innerhalb der Rubriken wiederholende Bestandteile sind in Komponenten ausgelagert.

Abbildung 7.6: Grundlegender Aufbau der Benutzerschnittstelle.

```
import streamlit as st
from Sections.Section import Section
from SiteComponents.ExerciseComponent import ExerciseComponent

class StudentSection(Section):

 def build_section(self):

  st.header("Software Engineering Übung")
  ...
  st.write("**Lernziel**")
  ...
   exercise_component = ExerciseComponent(theme_selectbox,
              complexity_selectbox, exercise_type_selectbox)
   ...
   exercise_component.display()
   ...

   upload_component = UploadComponent()
   upload_component.display()
```

Codebeispiel 7.4: Beispielhafte Implementierung der Klasse ‚StudentSection‘.

```
import streamlit as st

from SiteComponents.Component import Component

class ExerciseComponent(Component):

 def display(self):

  ...
  if self.__exercise_type_selectbox == "Durch Dozierende":
   ...
   self.__exercise_text = self.get_exercise_text_from_educator()

  html_text =
  '<br><h3>Generierte Aufgabe zu UML-Klassendiagrammen</h3><br>'
  + self.__exercise_text
  st.markdown(html_text, unsafe_allow_html=True)
  ...
```

Codebeispiel 7.5: Beispielhafte Implementierung der Klasse ‚ExerciseComponent‘.

Der Aufbau für ‚EducatorSection‘, ‚CollaborationSection‘ sowie ‚TutorialSection‘ ist vergleichbar mit der in Codebeispiel 7.4 enthaltenen Struktur und ist daher nicht explizit beschrieben.

7.4.4 Implementierung unterschiedlicher Plug-and-Play Komponenten

Die abstrakte Klasse ‚Component‘ aus dem Softwarepaket ‚SiteComponents‘ vererbt die Methode ‚display()‘ an ihre Kindklassen. Letztere unterscheiden sich in ihrem Verwendungszweck deutlich voneinander, folgen jedoch alle einem vergleichbaren Schema. Exemplarisch sind daher in Codebeispiel 7.5 respektive 7.6 die ‚ExerciseComponent‘ und die ‚UploadComponent‘ angeführt.

Bei Ersterer ist zu prüfen, ob eine Übungsaufgabe von Dozierenden erstellt wurde oder nicht. In ersterem Fall sind zusätzliche Inhalte, wie das empfohlene Vorgehen, zu visualisieren. Durch eine Indirektion über die Klasse ‚RequestHandler‘ ist eine auf dem Server gespeicherte Aufgabe abrufbar. Diese ist entweder über die ‚write(...)‘-Funktion von Streamlit oder, wie hier dargestellt, durch ein HTML-Markdown anzeigbar. Die zweite Option erlaubt die zusätzliche Angabe von HTML-Annotationen.

Die Abbildungen 7.7 und 7.8 verdeutlicht, wie eine grafische Darstellung dieser Komponenten in der Rubrik für Studierende aussieht.

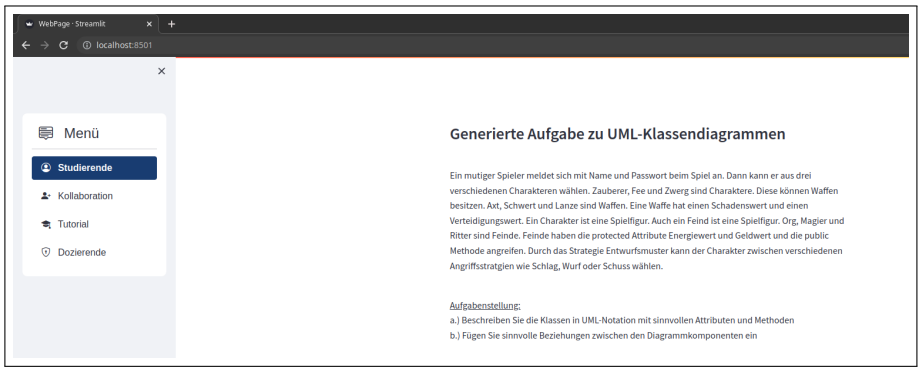


Abbildung 7.7: Darstellung der inkludierten Klasse ‚ExerciseComponent' innerhalb der Benutzerschnittstelle.

```
import streamlit as st
from SiteComponents.Component import Component

class UploadComponent(Component):

def display(self):

  self.__uploaded_file = st.file_uploader("",
                type=[".xml", ".png", ".jpg"],
                accept_multiple_files=False)
```

Codebeispiel 7.6: Beispielhafte Implementierung der Klasse ‚UploadComponent'.

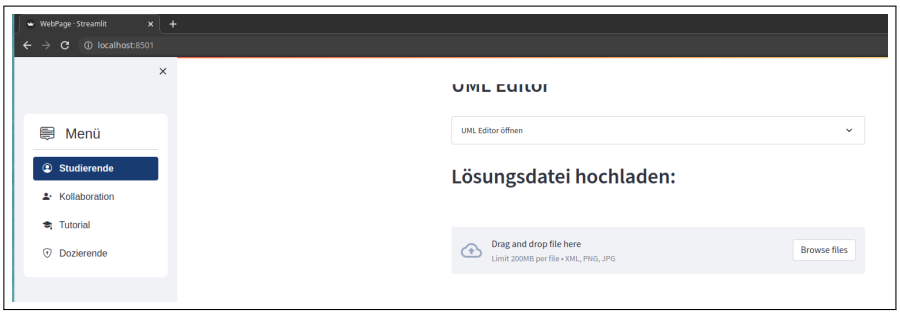


Abbildung 7.8: Darstellung der inkludierten Klasse ‚UploadComponent' innerhalb der Benutzerschnittstelle.

```
import requests

class RequestHandler:

 def __init__(self):

  self.__server_port = "4711"
  self.__server_url = "http://localhost" + ":" + self.__server_port


 def post(self, data, path ):

  url = self.__server_url + path
  response = requests.post(url, data=data)

  return response.text


 def get(self, data, path):

  url = self.__server_url + path
  response = requests.get(url, data=data)

  return response.text
```

Codebeispiel 7.7: Beispielhafte Implementierung der Klasse ‚RequestHandler‘.

7.4.5 Implementierung der Kommunikation zwischen Backend und Frontend

Die Kommunikation des Frontends zu dem Backend verläuft über die in Abbildung 7.2 dargestellte Klasse ‚RequestHandler‘. Die individuellen Rubriken sowie die Komponenten sind darüber hinaus in der Lage, Informationen abzurufen und zu versenden. In Codebeispiel 7.7 wird der Aufbau dieser Klasse deutlich. Innerhalb des Konstruktors sind zwei private Attribute für den benötigten Server-Port respektive die Server-URL definiert. Für die HTTP-Operationen GET und POST gibt es jeweils eine Methode. Diese nehmen Daten im JSON-Format und einen eindeutigen Pfad entgegen, der in Kombination mit der Server-URL spezifiziert, welche Server-Methode anzusprechen ist.

Abbildung 7.9 gibt einen Einblick über die Kommunikation des Frontends mit dem Backend. In der Rubrik Dozierende wurde für die Erstellung des Schaubilds eine Übungsaufgabe hinzugefügt und dann über einen Klick auf den Button „Aufgabe bereitstellen“ (in der Abbildung durch eine rote 1. kenntlich gemacht) an das Backend gesendet. Das dargestellte Konsolenfenster zeigt dessen darauffolgende Ausgabe (mit einer roten 2. kenntlich

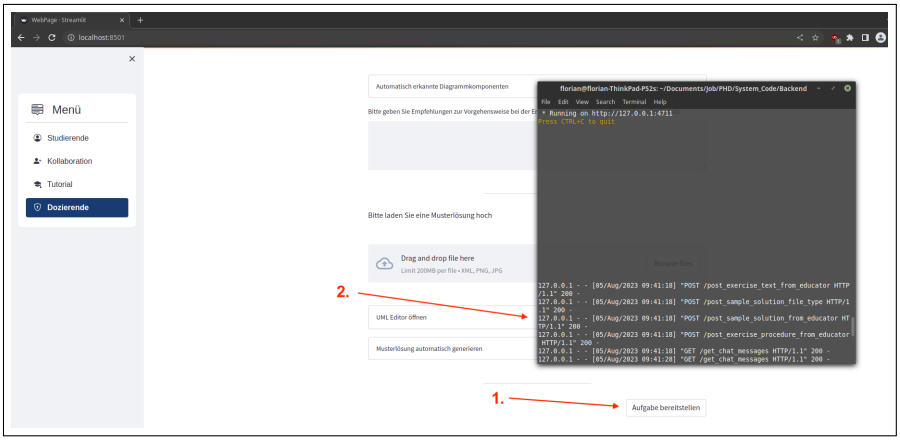


Abbildung 7.9: Darstellung der Kommunikation zwischen Frontend und Backend.

gemacht). Somit ist belegbar, dass der Nachrichtenaustausch zwischen den zwei voneinander unabhängigen Entitäten möglich ist.

7.5 Darstellung der im Frontend verfügbaren Rubriken

Im Anschluss an die Implementierung ist eine Darstellung der in den Tabellen 7.2, 7.3, 7.4 und 7.5 angeführten Inhalte möglich. Die folgenden Abschnitte zeigen mit ihren Abbildungen die vorgestellten Rubriken. Da diese bereits durch die Konzeption sowie die Tabellen umfassend erläutert sind, wird an dieser Stelle davon abgesehen.

7.5.1 Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle

Die Abbildungen 7.10, 7.11, 7.12, 7.13, 7.14, 7.15 sowie 7.16 beinhalten die grafische Darstellung der Rubrik für Dozierende. Die Inhalte sind durch den Autor zu Demonstrationszwecken hinzugefügt.

Zu erwähnen ist, dass eine breitere Darstellung des UML-Editors und somit ein effizientes Arbeiten mit diesem möglich ist.

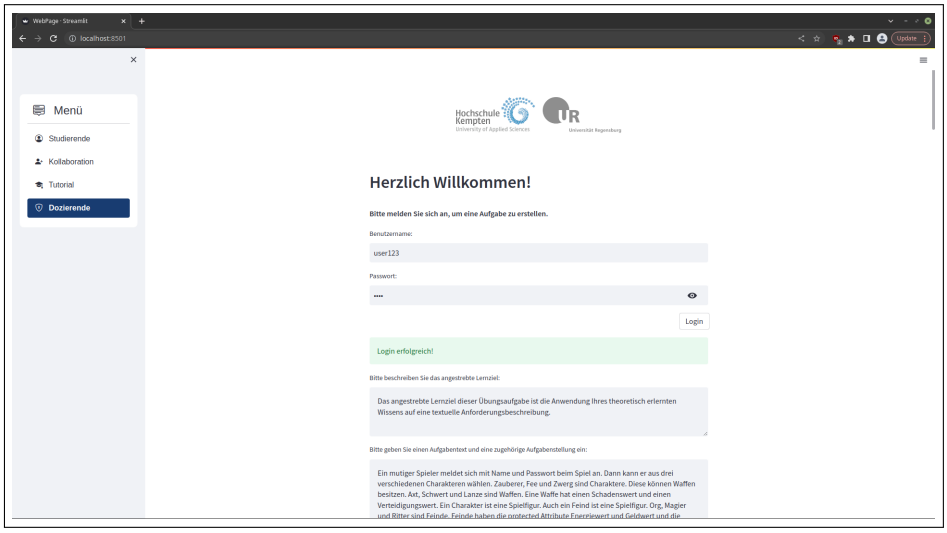


Abbildung 7.10: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 1.

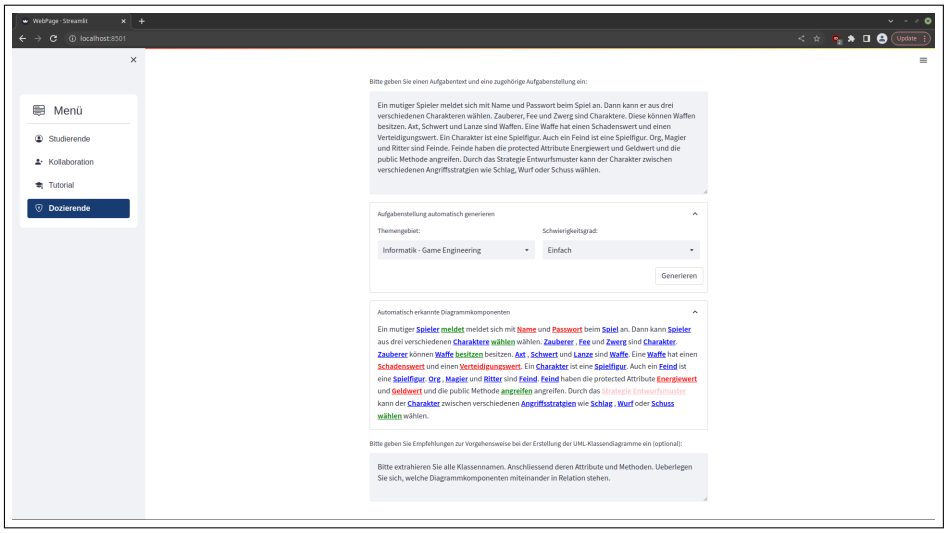


Abbildung 7.11: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 2.

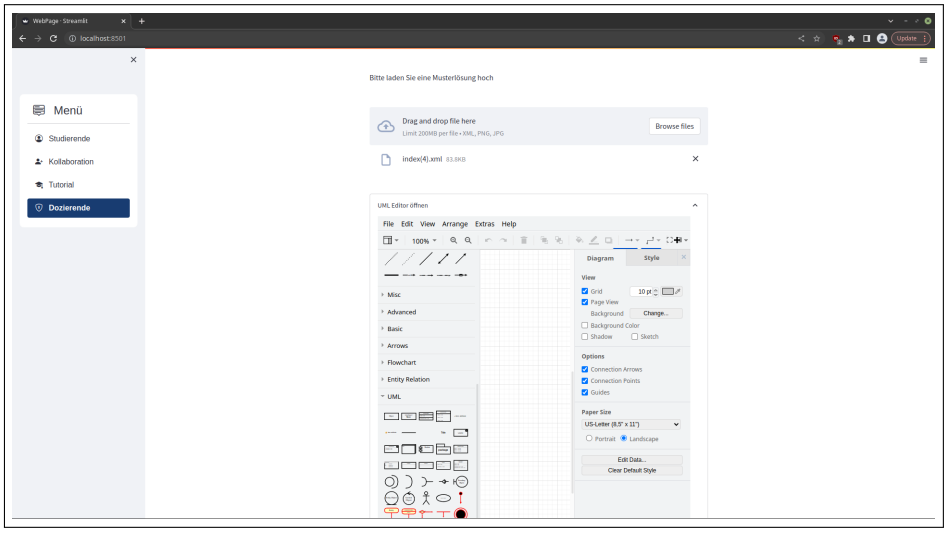


Abbildung 7.12: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 3.

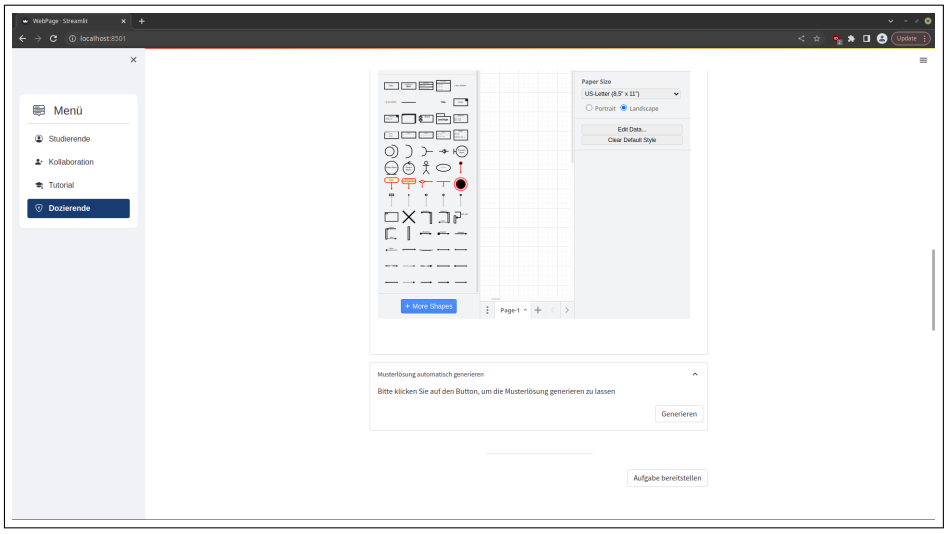


Abbildung 7.13: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 4.

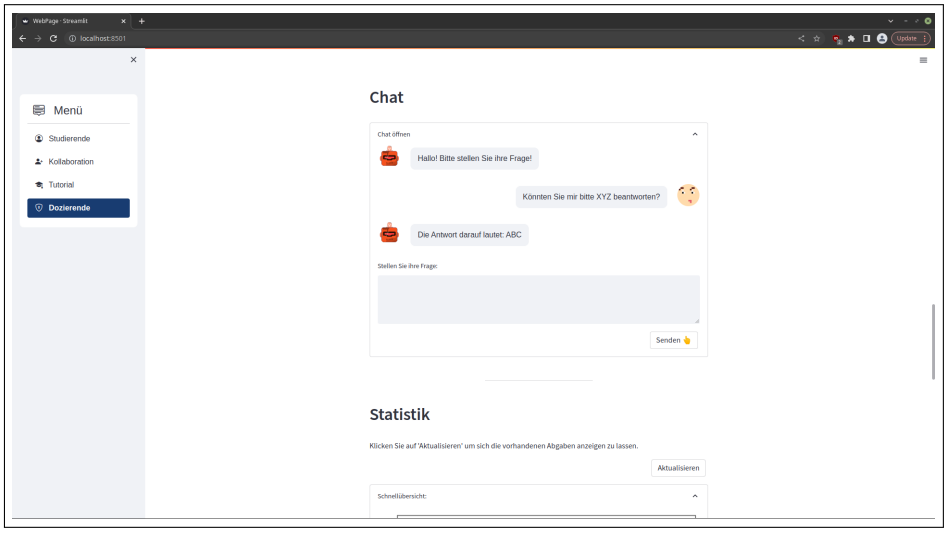


Abbildung 7.14: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 5.

Abbildung 7.15: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 6.

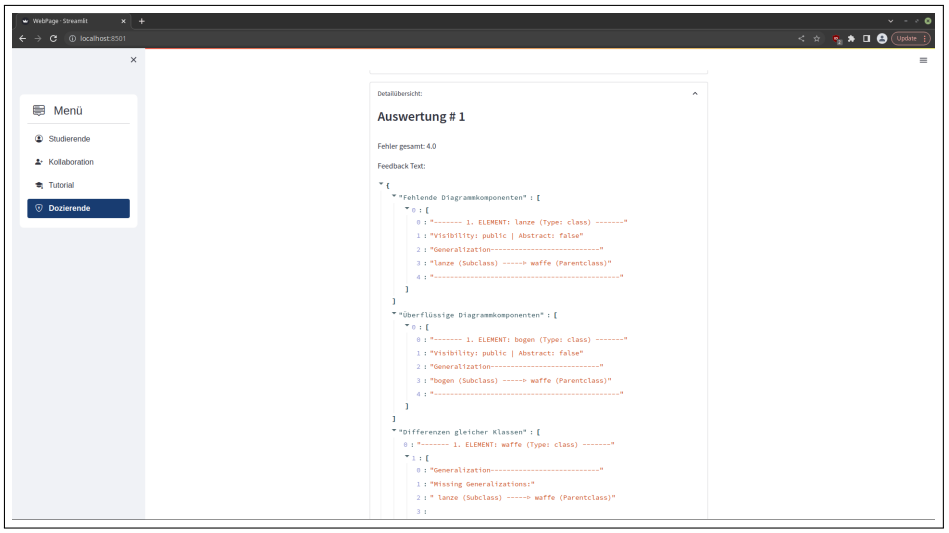


Abbildung 7.16: Darstellung der Rubrik für Dozierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 7.

7.5.2 Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle

Die Abbildungen 7.17, 7.18, 7.19, 7.20, 7.21, 7.22, 7.23, 7.24 sowie 7.25 beinhalten die grafische Darstellung der Rubrik für Studierende. Die Inhalte sind durch den Autor zu Demonstrationszwecken hinzugefügt.

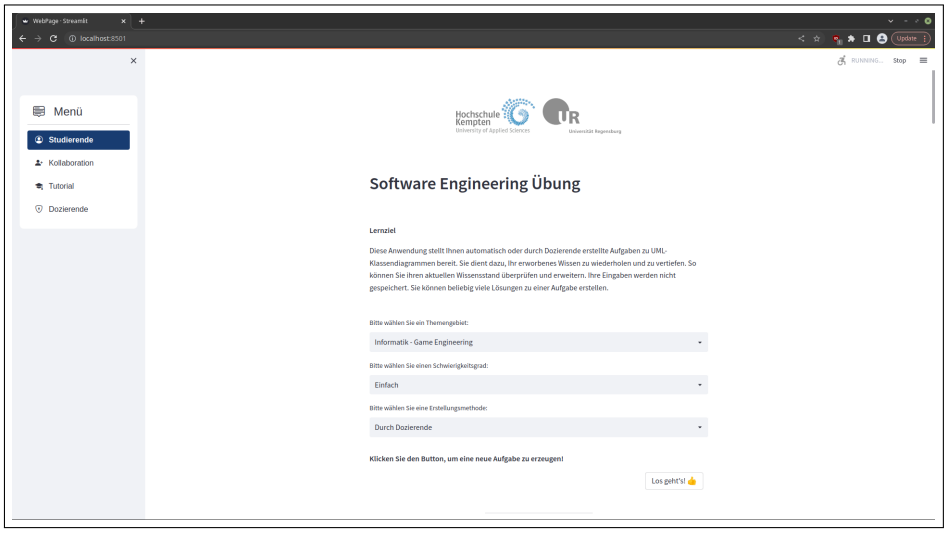


Abbildung 7.17: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 1.

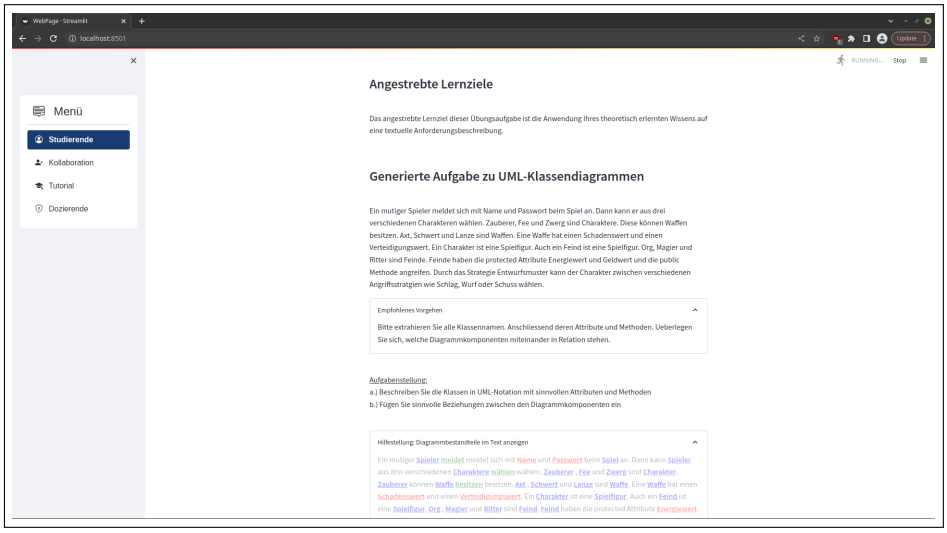


Abbildung 7.18: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 2.

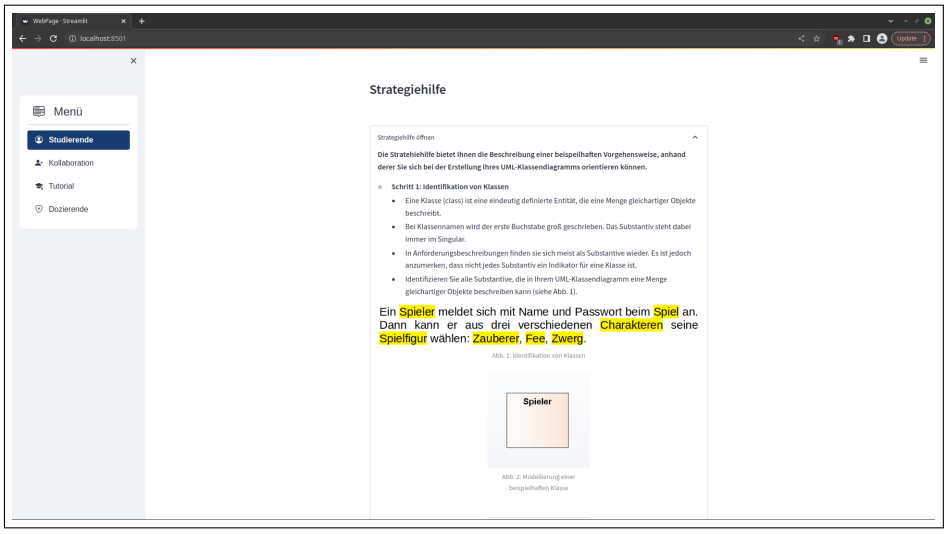


Abbildung 7.19: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 3.

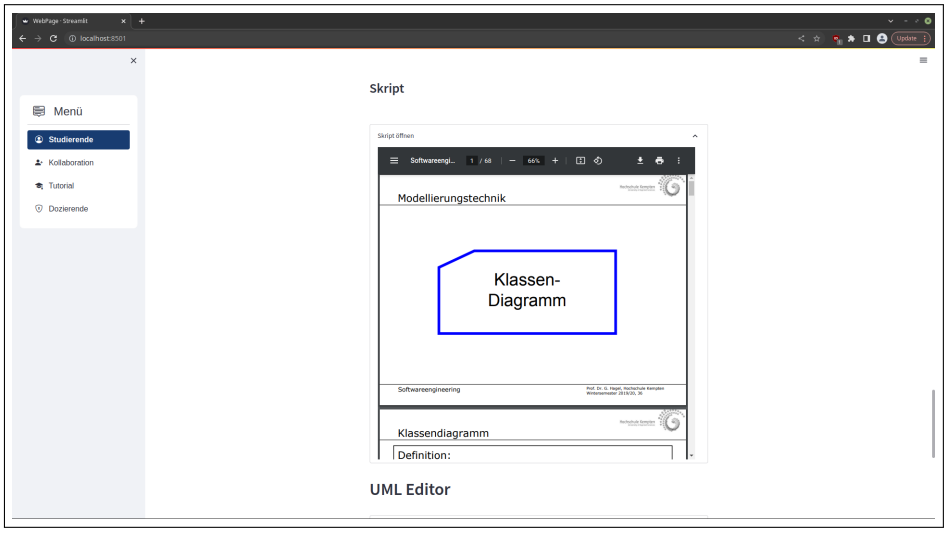


Abbildung 7.20: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 4.

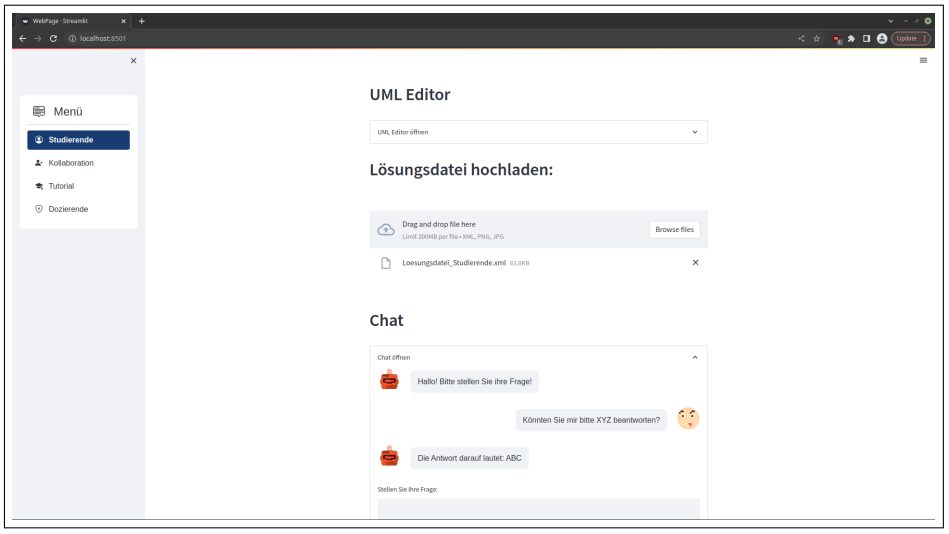


Abbildung 7.21: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 5.

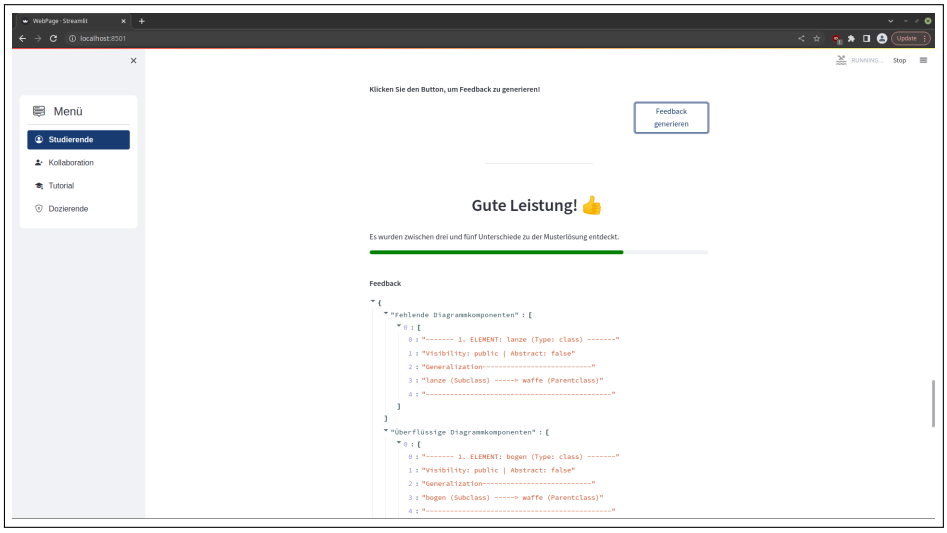


Abbildung 7.22: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 6.

Abbildung 7.23: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 7.

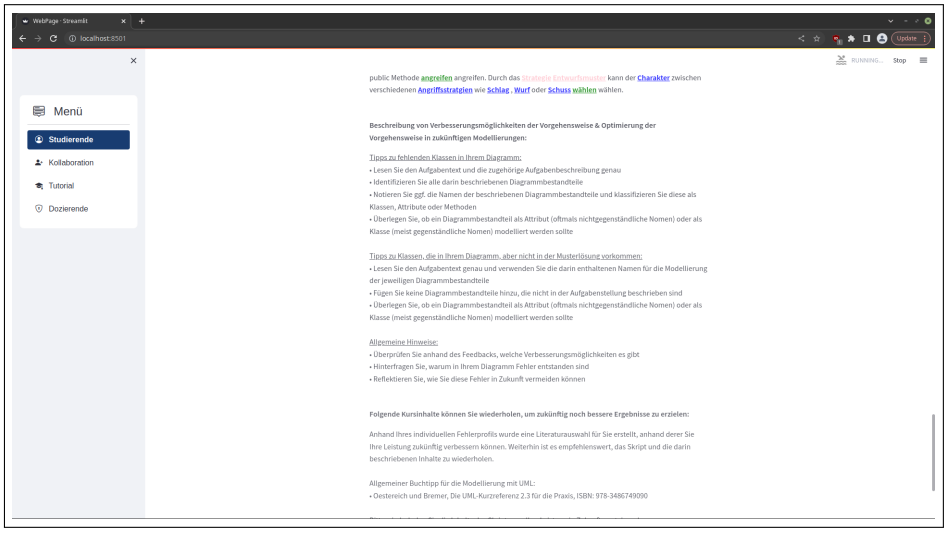


Abbildung 7.24: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 8.

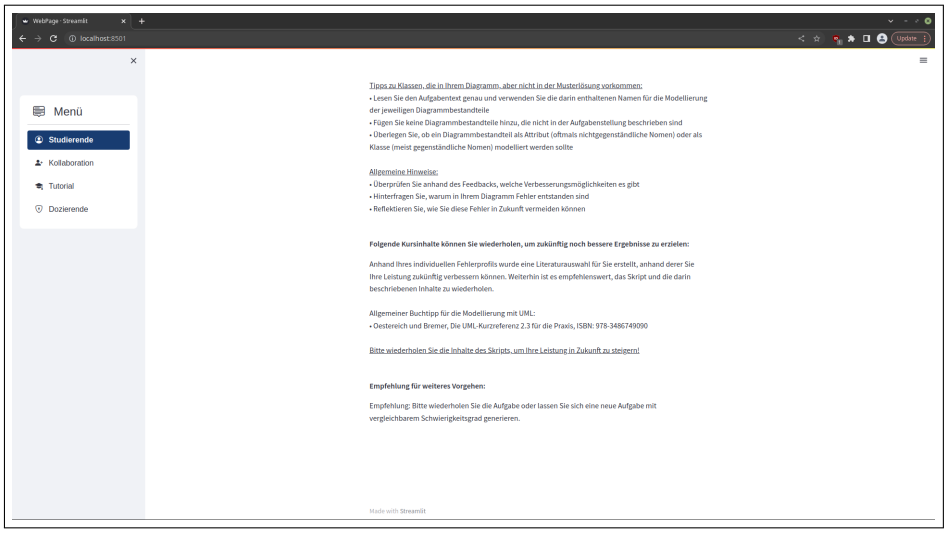


Abbildung 7.25: Darstellung der Rubrik für Studierende innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 9.

7.5.3 Darstellung der Rubrik für das Tutorial innerhalb der Benutzerschnittstelle

Die Abbildungen 7.26, 7.27, 7.28 sowie 7.29 beinhalten die grafische Darstellung der Rubrik für das Tutorial. Die Inhalte sind durch den Autor zu Demonstrationszwecken hinzugefügt.

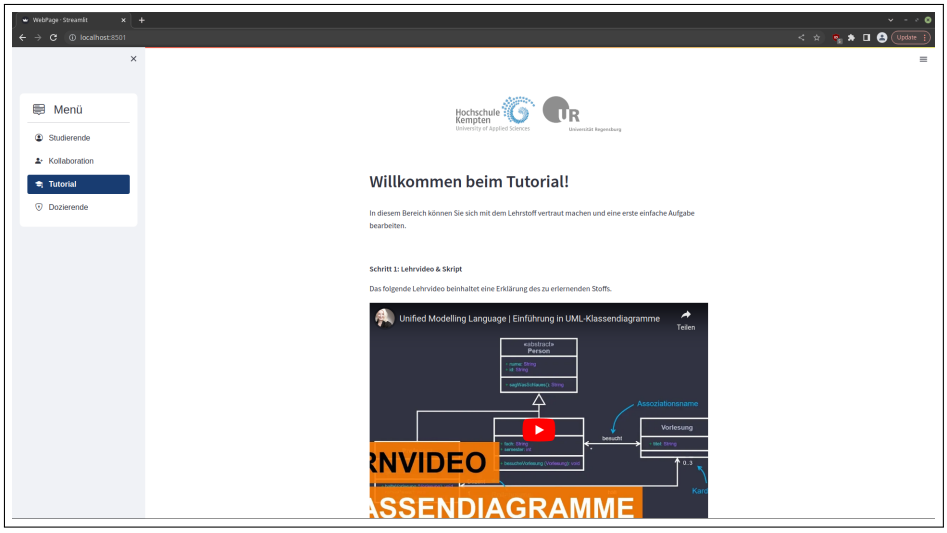


Abbildung 7.26: Darstellung der Rubrik für das Tutorial innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 1.

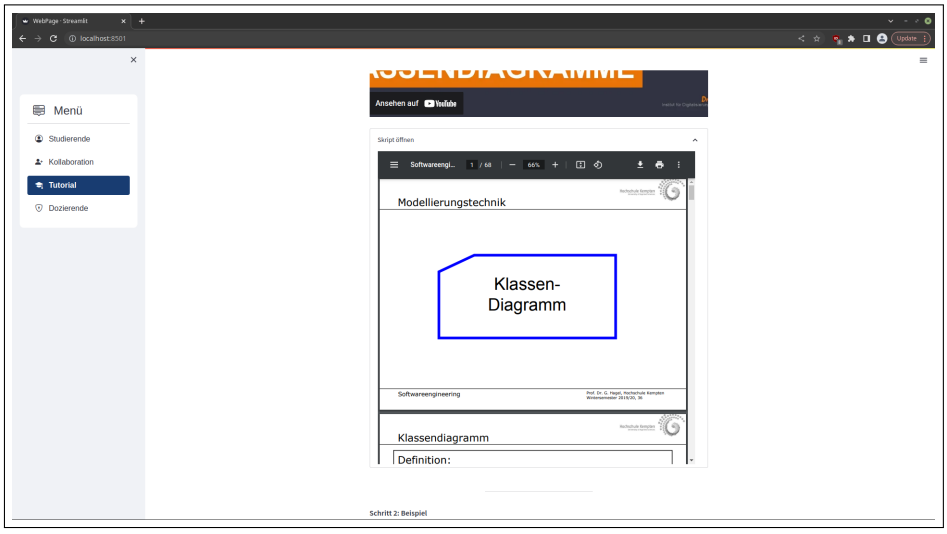


Abbildung 7.27: Darstellung der Rubrik für das Tutorial innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 2.

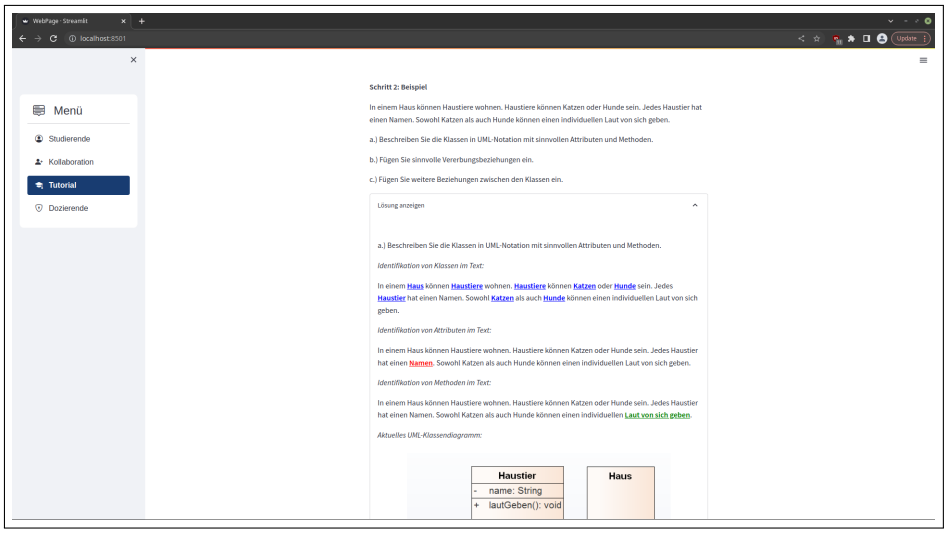


Abbildung 7.28: Darstellung der Rubrik für das Tutorial innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 3.

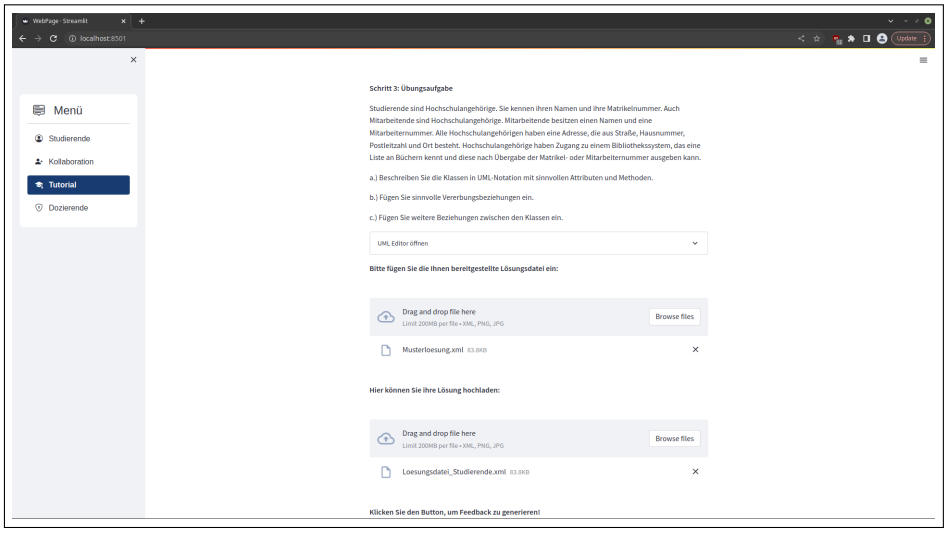


Abbildung 7.29: Darstellung der Rubrik für das Tutorial innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 4.

7.5.4 Darstellung der Rubrik für die Kollaboration innerhalb der Benutzerschnittstelle

Die Abbildungen 7.30 sowie 7.31 beinhalten die grafische Darstellung der Rubrik für das Tutorial. Die Inhalte sind durch den Autor zu Demonstrationszwecken hinzugefügt.

Das Feedback für die abgebildete Übungsaufgabe ist nicht explizit aufgezeigt, da dieses bereits in der Rubrik für Studierende einsehbar ist.

Abbildung 7.30: Darstellung der Rubrik für die Kollaboration innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 1.

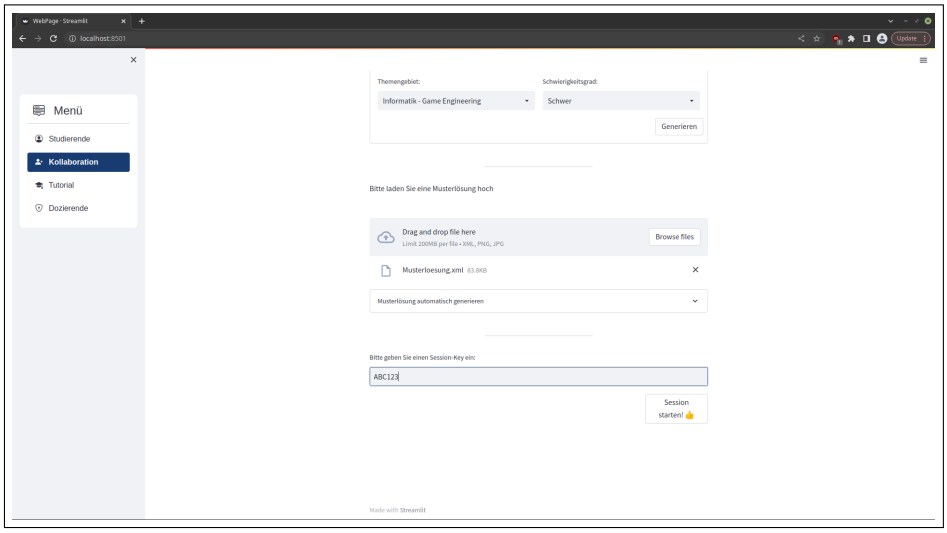


Abbildung 7.31: Darstellung der Rubrik für die Kollaboration innerhalb der Benutzerschnittstelle im Anschluss an die Implementierung - Teil 2.

7.6 Beantwortung der zentralen Forschungsfrage

Dieses Kapitel beinhaltet eine Konzeption sowie eine prototypische Implementierung eines computergestützten Systems für die Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering. Die bis dato erarbeiteten theoretischen Grundlagen sowie Resultate finden hier Anwendung.

Das Ergebnis stellen ein Server respektive eine interaktive Webseite dar. Diese folgen den REST-Kriterien und sind somit auch unabhängig voneinander einsetzbar. In Kombination ermöglichen sie die digitale Abbildung einer Übungseinheit, in der Dozierende ihren Studierenden Aufgaben stellen oder Studierende einzeln sowie in Kollaboration miteinander arbeiten können. Für die Aneignung des benötigten Wissens steht Lernenden die Rubrik Tutorial zur Verfügung. Sie beinhaltet neben Lehrinhalten auch einfache Beispielaufgaben (teils mit vorgegebenem Lösungsweg und dem gewünschten Endresultat).

Das System stellt einen ganzheitlichen Ansatz für die Lehre von UML-Klassendiagrammen dar, da nicht nur individuelle Herausforderungen von Studierenden, sondern der gesamte Lernprozess abbildbar ist. Neben weiteren Kriterien unterschiedet sich die präsentierte Anwendung von den in der Wissensbasis enthaltenen. Die in Kapitel 3 anhand bestehender Fachliteratur sowie Befragungen der Zielgruppen erhobenen Anforderungen lassen sich durch den bereitgestellten Prototyp erfüllen. Eine Exklusion einer solchen geschah nur mit triftigem Grund. Die zentrale Forschungsfrage der Arbeit lässt sich folglich durch dieses Kapitel beantworten.

KAPITEL 8

Evaluation

Bei dem konzipierten und prototypisch implementierten System für die Unterstützung der hochschulischen Lehre von UML-Klassendiagrammen handelt es sich um einen Ansatz für studierendenzentriertes Lehren und Lernen. Die erhobenen Anforderungen von und für Dozierende zu realisieren, ist ein wichtiger Bestandteil dieses Vorhabens. Die Entwicklung dient jedoch primär vor allem der Verbesserung des Lernerfolgs von Studierenden. Die Evaluation fokussiert sich folglich auf eine Auswertung des Systems mit dieser Hauptzielgruppe.

Die Untersuchung des Artefakts anhand der Umgebung ist nun als letzter Schritt dieser Arbeit vorgesehen. Die von Hevner u. a. (2004) erläuterten Zyklen wurden im Sinne des DRS-Forschungsdesigns mehrfach durchlaufen. Für eine Auswertung gibt es multiple Möglichkeiten und Vorgehensweisen. Die Wahl fiel in diesem Kapitel auf eine quantitative Studie mithilfe eines Fragebogens. Shneiderman u. a. (2017) zufolge eignet sich diese Form der Evaluation in diesem Kontext sehr gut, da die Antworten bei geringen Teilnehmerzahlen, wie beispielsweise bei leitfadengestützten Experteninterviews, stärker variieren können (obwohl die Autoren ihre Aussage auf die Auswertung grafischer Benutzerschnittstellen beziehen, ist der Gedanke an dieser Stelle übertragbar).

Auch eine Vielzahl unterschiedlicher Aspekte des Prototyps, wie bei-

spielsweise die Benutzerschnittstelle, können evaluiert werden. Dafür ließe sich exemplarisch der Questionnaire for User Interaction Satisfaction heranziehen (vgl. Chin u. a., 1988; vgl. Shneiderman u. a., 2017). Bei dem Prototyp handelt es sich bis dato um ein experimentelles Softwaresystem, welches für den beschriebenen Problemkontext innovative Lösungsansätze in sich vereint. Etablierte Fragebögen wie der Questionnaire for User Interaction Satisfaction sind daher oftmals nicht ausreichend geeignet, um beispielsweise die Benutzerschnittstelle zu evaluieren. Darin enthaltene Fragen nach der bisherigen Nutzungsdauer oder der Verständlichkeit der Bedienungsanleitung lassen sich in diesem experimentellen Stadium der computergestützten Hilfestellung von potenziellen Anwender*innen nicht beantworten (vgl. Chin u. a., 1988; vgl. Shneiderman u. a., 2017). Weiterhin ist es das Ziel der quantitativen Untersuchung, unterschiedliche Aspekte des Prototyps auszuwerten. Die Evaluation konzentriert sich folglich nicht nur auf die Benutzerschnittstelle, sondern beispielsweise auch auf das bereitgestellte Feedback. Einige Bestandteile aus dem Questionnaire for User Interaction Satisfaction finden jedoch in kondensierter Form Einzug in den entwickelten Fragebogen. Die im Folgenden angeführte Studie weist aus den genannten Gründen ebenfalls experimentellen Charakter auf und versucht, die beschriebenen Aspekte miteinander zu vereinen.

8.1 Aufbau der Studie

Die Durchführung der Studie erfolgte im Sommersemester 2023 an der Hochschule für angewandte Wissenschaften in Kempten. Als potenzielle Teilnehmende kamen Studierende in einem informatiknahen Studiengang infrage. Weiterhin müssen sie den Software Engineering-Kurs mit den zugehörigen Übungen besucht haben. Ein spezielles Fachsemester ist dabei nicht vorgegeben. Da es sich bei dem Prototyp um ein studierendenzentriertes Softwaresystem handelt, soll er speziell mit dieser Gruppe evaluiert werden.

In Anlehnung an die von Huber und Hagel (2022a) entwickelte Studie dient bestehende Fachliteratur (vgl. Chin u. a., 1988; vgl. Porst, 2014; vgl. Döring und Bortz, 2016; vgl. Shneiderman u. a., 2017; vgl. Lewis, 2018; vgl. Baur und Blasius, 2019) als Ausgangsbasis für den Aufbau des folgenden Fragebogens. Dieser ist in sieben Bereiche untergliedert, welche die für Stu-

dierende relevantesten Aspekte des Systems abdecken. Zu nennen sind allgemeine Informationen über die Partizipierenden, ihre Meinung zu der Benutzeroberfläche, den für Lernende zugänglichen Rubriken, dem erhaltenen Feedback sowie eine allgemeine Befragung zu den präsentierten Inhalten. Für jeden dieser Bestandteile sind mehrere geschlossene Fragen, gefolgt von einer Gesamtbewertung anhand einer Skala und zwei offenen Fragen vorgesehen (vgl. Porst, 2014). Von einer zusätzlichen Anführung dieser wird an dieser Stelle bewusst Abstand genommen, da sich der Fragebogen über neun Seiten erstreckt. Sie sind jedoch in Anhang A.3 und im folgenden Abschnitt bei der Auswertung der Ergebnisse einsehbar.

Als Antwortmöglichkeiten auf die geschlossenen Fragen gab es die im Anschluss angeführten Optionen (in Anlehnung an (vgl. Huber und Hagel, 2022a). Übersetzung durch den Autor):

- Ich stimme nicht zu.
- Ich stimme eher nicht zu.
- Ich stimme eher zu.
- Ich stimme voll zu.
- Kann ich nicht beurteilen.

Insgesamt konnten 14 Studierende aus unterschiedlichen Semestern und Studiengängen für eine Teilnahme gewonnen werden. Auf drei Termine und somit in Gruppen verteilt, fand eine Präsentation des Systems vor Ort oder mittels der Videokonferenzsoftware Zoom[37] statt. Während der Präsentation hatten die Teilnehmenden die Möglichkeit, Fragen zu stellen und einzelne Funktionalitäten selbst zu testen. Der zeitliche Rahmen bewegte sich zwischen 60 Minuten und 90 Minuten. Gerade Studien mit Fragebögen limitieren die Anzahl möglicher Teilnehmenden oftmals nicht. Dies gilt auch für die vorliegende Auswertung. Aufgrund des zeitlichen Rahmens war es jedoch nicht möglich, eine größere Gruppe an Studierenden für eine Teilnahme zu begeistern.

[37]Einsehbar unter: https://zoom.us/de

8.2 Ergebnisse der Studie

Die sieben genannten Aspekte, bezogen auf die Studierenden, sind durch die nachfolgenden Unterkapitel abgedeckt und spiegeln somit die Struktur des Fragebogens wieder. Anhand des Umfangs wurde durch den Autor versucht, ein möglichst umfassendes Bild über das konzipierte System zu erhalten.

Es ist anzumerken, dass die Studierenden in wenigen Ausnahmefällen eine einzelne geschlossene Frage nicht beantwortet haben. Bei deren Auswertung ist dies entsprechend vermerkt. Weithin kam es bei den Antworten auf die offenen Fragen zu Wiederholungen, die es entsprechend zu konsolidieren galt. Einige Studierende haben dabei auch Antworten auf die beiden unterschiedlichen Fragestellungen in eines der zwei vorgesehenen Felder geschrieben. Durch den Autor fand in diesem Fall eine entsprechende Separierung statt. Einige der Ergebnisse sind (sofern sinnvoll) prozentual in Form von Dezimalzahlen mit einer Nachkommastelle angeführt. Aufgrund von Ungenauigkeiten durch Rundungen ist es möglich, dass die Summe der Ergebnisse nicht überall exakt 100% beträgt.

8.2.1 Demografische Informationen zu den partizipierenden Studierenden

In einem ersten Schritt sind einige persönliche Angaben der Studierenden zu erheben. Diese sind jedoch keinesfalls auf individuelle Personen rückführbar und nehmen keinen direkten Einfluss auf die Evaluation. Sie sollen lediglich verdeutlichen, dass die Gruppe der Partizipierenden aus Studierenden unterschiedlichen Alters und Geschlechts aus diversen informatiknahen Studiengängen besteht.

- Wie alt sind Sie?
  Die befragten Studierenden waren zwischen 20 und 31 Jahre alt. Das durchschnittliche Alter betrug 25.6 Jahre.

- Bitte geben Sie ihr Geschlecht an.
  Drei der Studierenden waren weiblich, 11 männlich. Unter den Teilnehmenden wählte niemand die Antwortmöglichkeiten „Divers“ oder „Keine Antwort“.

- Bitte geben Sie Ihren Studiengang an.
  Sieben Studierende gaben an, Wirtschaftsinformatik zu studieren. Wohingegen sechs den Studiengang Informatik und eine weitere Person Informatik - Game Engineering angaben.

Aufgrund der Diversität der Teilnehmenden ist darauf zu schließen, dass eine umfassende und vielfältige Rückmeldung erhalten werden kann.

8.2.2 Allgemeine Evaluation zu der Benutzeroberfläche

Das Frontend in Form einer grafischen Benutzeroberfläche ist der Bestandteil des Systems, mit dem die Akteure interagieren können. Es ist daher wichtig zu untersuchen, ob sie deren Ansprüchen genügt.

Die folgende Aufzählung bildet alle zu diesem Themenschwerpunkt gestellten, geschlossenen Fragen inklusive eines Überblicks der zugehörigen Antworten ab:

- Die grafische Gestaltung der Benutzeroberfläche empfinde ich als ansprechend.
  Die Studierenden stimmten dieser Aussage zu 57,1% eher und zu 42,9% voll zu.

- Die Benutzeroberfläche finde ich übersichtlich.
  Die Studierenden empfanden die Benutzeroberfläche zu 28,6% eher und zu 71,4% sehr ansprechend.

- Innerhalb der Benutzeroberfläche finde ich mich intuitiv zurecht.
  Die Teilnehmenden fanden sich zu 21,4% eher und zu 71,4% sehr gut zurecht. Eine partizipierende Person (7,1%) gab an, dies nicht beurteilen zu können.

- Die Benutzeroberfläche nehme ich als selbsterklärend wahr.
  64,3% der Studierenden stimmten eher und 33,7% voll zu.

- Die Benutzeroberfläche bewerte ich für den Einsatz in der hochschulischen Lehre als angemessen.
  Fast alle Teilnehmenden stimmten dieser Aussage voll (92,9%) und lediglich eine Person eher (7,1%) zu.

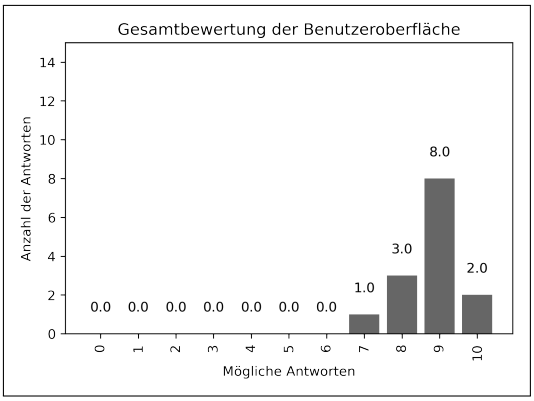


Abbildung 8.1: Darstellung der Gesamtbewertung der grafischen Benutzeroberfläche im Rahmen einer Evaluation.

- Den strukturellen Aufbau der Benutzeroberfläche bewerte ich als gut. Die Studierenden gaben zu 92,9% an, der Aussage voll zuzustimmen. 7,1% stimmten eher zu.

Wie die Auswertung demonstriert, bewerten die Studierenden die Benutzeroberfläche als übersichtlich, ansprechend und intuitiv bedienbar. Weiterhin sind sie der Ansicht, dass sie sich gut für den Einsatz in der hochschulischen Lehre eignet. Die in Abbildung 8.1 dargestellte Gesamtbewertung unterstreicht dieses Ergebnis. Die Teilnehmenden konnten dabei eine Bewertung auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, vornehmen. Einmal (7.1%) fiel die Wahl auf eine 7, wohingegen drei Personen (21,4%) 8, acht Personen 9 (64,3%) und zwei Personen (14,3%) 10 Punkte vergaben.

Die erste offene Frage evaluierte, ob Erweiterungswünsche bezüglich der Funktionalität aufkamen. Sie finden sich stichpunktartig und konsolidiert in der folgenden Auflistung:

- Legende für die Farbgebung der zu erkennenden Diagrammkomponenten.
- Beschreibung, wer Chat-Nachrichten lesen und empfangen kann.
- Darstellung der Benutzeroberfläche in einem dunklen Design (Hinweis durch den Autor: diese Funktionalität steht bereits zur Verfügung, ist jedoch bei der Evaluation nicht explizit demonstriert worden).
- Einsatz von mehr Farbgebung.

Zusätzlich gab es weiteres Feedback seitens der Teilnehmenden:

- Intuitiv gestaltet.
- Man findet sich ohne große Einweisung zurecht.
- Ansprechend und schlicht gestaltet (dadurch ist eine einfache Navigation möglich).
- Strukturelle Trennung in Bereiche ist sinnvoll.
- Einsatz von Smileys ist sehr ansprechend.
- Als Lernumgebung sehr gut geeignet.

8.2.3 Evaluation der Rubrik für das Tutorial

Die Rubrik für das Tutorial ist den Studierenden als erstes präsentiert worden. Sofern sie in ihrem Lernprozess noch am Anfang stehen und sich neues Wissen aneignen möchten, kann dieses einen Einstiegspunkt darstellen.

Die folgenden Antworten gaben die Befragten zu diesem Themenschwerpunkt:

- Die Inhalte des Programmbereichs Tutorial bewerte ich als lehrreich.
  21,4% der Studierenden stimmten dieser Aussage eher und 78,6% voll zu.
- Den strukturellen Aufbau des Programmbereichs Tutorial bewerte ich als sinnvoll.
  Den strukturellen Aufbau der Rubrik bewerteten 21,4% eher und 78,6% sehr sinnvoll.
- Ich kann mir vorstellen, den Programmbereich Tutorial für den Aufbau bzw. die Erweiterung meines Wissens zu verwenden.
  Dieser Aussage pflichteten die Teilnehmenden zu 28,6% eher und zu 71,4% voll bei.
- Die beispielhafte Übungsaufgabe empfinde ich als hilfreich.
  21,4% der Studierenden empfanden die Übungsaufgabe als eher und 71,4% als sehr hilfreich. Eine Person (7,1%) gab an, dies nicht beurteilen zu können.

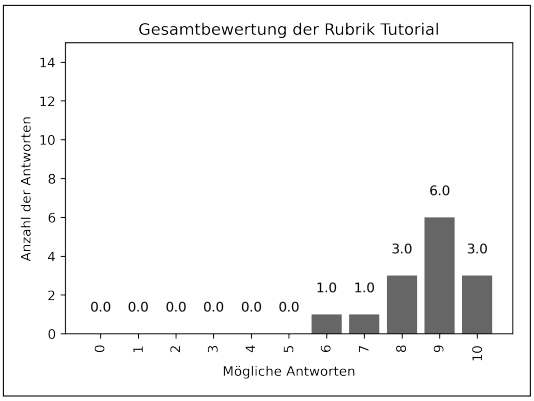


Abbildung 8.2: Darstellung der Gesamtbewertung der Rubrik für das Tutorial im Rahmen einer Evaluation.

- Die beispielhafte Lösung für die bereitgestellte Übungsaufgabe empfinde ich als hilfreich.
  Die Teilnehmenden stimmten der Aussage zu 21,4% eher und zu 71,4% voll zu. Von einer Person (7,1%) gab es keine Antwort.

- Den Schwierigkeitsgrad der dargestellten Übungsaufgabe bewerte ich als angemessen.
  Bezogen auf den Schwierigkeitsgrad der dargestellten Übungsaufgabe stimmten 7,1% eher nicht, 35,7% eher und 50,0% voll zu. Eine weitere Person (7,1%) gab an, dies nicht beurteilen zu können.

- Die dargestellte Übungsaufgabe empfinde ich als hilfreich.
  Die Teilnehmenden bewerteten die Übungsaufgabe zu 42,9% als eher und zu 50,0% als sehr hilfreich. Eine weitere Person (7,1%) gab an, dies nicht beurteilen zu können.

Die von den Studierenden abgegebenen Gesamtbewertungen sind als sehr positive Rückmeldung für diese Rubrik zu interpretieren. Wie Abbildung 8.2 aufzeigt, vergaben sie je einmal 6 und 7 (je 7,1%), dreimal 8 (21,4%), sechsmal 9 (64,3%) und dreimal 10 (21,4%) Punkte. Dieses Ergebnis deckt sich mit den Antworten der geschlossenen Fragen. Die Teilnehmenden empfinden diesen Programmbereich als inhaltlich sinnvoll und strukturell angemessen.

Als zusätzliche Erweiterungswünsche kamen die folgenden Punkte auf:

- Noch mehr Beispiele und Tipps.

- Eine Einstellung für den Umfang der textuellen Anforderungsbeschreibung.
- Eine sehr komplexe Beispielaufgabe für Übungszwecke.
- Mehr Lehrvideos zum Thema UML-Klassendiagramme.

Bei der offenen Frage nach weiterem Feedback meldeten die Studierenden lediglich zurück, dass das Tutorial eine gute Einweisung in die Thematik darstelle.

8.2.4 Evaluation zu dem Programmbereich für Studierende

Die Rubrik für Studierende wurde den Teilnehmenden als zentraler Bestandteil des Systems präsentiert. Dabei erhielten sie einen Einblick in alle Funktionalitäten und mögliche Szenarien. Im Rahmen der Evaluation gab der Prototyp bei einer automatisierten Generierung immer eine identische textuelle Anforderungsbeschreibung aus und erstellte somit auch jedes Mal dieselbe Musterlösung. Zwei durch den Autor vorgefertigte studentische UML-Klassendiagramme dienten als Vergleichsobjekte. Erstere enthielt nur einen Fehler. Letztere stimmte nicht im entferntesten mit der Musterlösung überein und erforderte daher eine Ausgabe aller verfügbaren Tipps und Hilfestellungen. Mit der Zurverfügungstellung einer identischen Übungsaufgabe für alle Partizipierenden wurde versucht, vergleichbare Bedingungen während des Evaluationsprozesses zu schaffen.

Für die geschlossenen Fragen zu diesem Themenschwerpunkt gaben die Partizipierenden folgende Antworten:

- Den strukturellen Aufbau des Programmbereichs Studierende bewerte ich als sinnvoll.
  Zu 28,6% stimmten die Studierenden dieser Aussage eher und zu 64,3% voll zu. Eine Person (7,1%) hat keinerlei Angaben gemacht.
- Die Beschreibung des Lernziels enthält meiner Meinung nach alle wichtigen Aspekte.
  Die Teilnehmenden waren zu 21,4% eher und zu 64,3% absolut der

Meinung, dass das Lernziel alle wichtigen Aspekte enthält. Zwei Personen (14,3%) gaben an, dies nicht beurteilen zu können.

- Die verfügbaren Themengebiete für die automatisierte Erstellung einer Übungsaufgabe bewerte ich als ausreichend.
  Die verfügbaren Themengebiete sind zu 42,9% als eher und zu 50,0% als vollkommen ausreichend bewertet worden. Eine Person (7,1%) gab an, dies nicht beurteilen zu können.

- Die verfügbaren Schwierigkeitsgrade für die automatisierte Erstellung einer Übungsaufgabe bewerte ich als ausreichend.
  14,3% der Studierenden stimmten eher und 71,4% voll zu. Zwei Personen (14,3%) gaben an, dies nicht beurteilen zu können.

- Die Hilfestellung, welche die Diagrammbestandteile einer textuellen Anforderungsbeschreibung anzeigt, bewerte ich als gut verständlich.
  Neben einer Person (7,1%), die eher zustimmte, gaben 85,7% an, die Hilfestellung als sehr verständlich zu bewerten. Eine Person (7,1%) gab an, dies nicht beurteilen zu können.

- Die Inhalte der angezeigten Strategiehilfe empfinde ich als hilfreich.
  Die Inhalte der Strategiehilfe wurden zu 28,6% als eher und zu 64,3% als sehr hilfreich eingestuft. Eine Person (7,1%) gab an, dies nicht beurteilen zu können.

- Die Anzeige des Vorlesungsskripts empfinde ich als hilfreich.
  Zu jeweils 14,3% stimmten die Teilnehmenden dieser Aussage eher nicht und eher zu. 64,3% bewerteten diese als sehr hilfreich, während eine Person (7,1%) angab, dies nicht beurteilen zu können.

- Das Hochladen eines UML-Klassendiagramms hat gut funktioniert.
  42,9% der Studierenden stimmten voll zu, wohingegen 57,1% angaben, dies nicht beurteilen zu können.

- Die Chatfunktion empfinde ich als hilfreich.
  Die Chatfunktion wurde zu 7,1% als eher nicht, zu 28,6% als eher und zu 35,7% als sehr hilfreich bewertet. Weitere 28,6% gaben an, dies nicht beurteilen zu können.

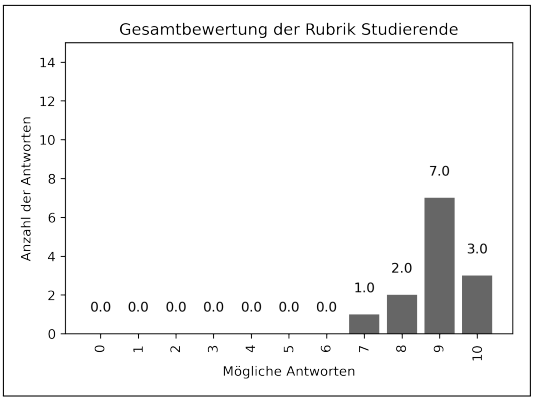


Abbildung 8.3: Darstellung der Gesamtbewertung der Rubrik für Studierende im Rahmen einer Evaluation.

Zusammenfassend lässt sich sagen, dass die vorgestellte Rubrik von den Teilnehmenden als durchaus positiv bewertet wurde. Neben einem sinnvollen strukturellen Aufbau empfinden die Studierenden die Inhalte als sinnvoll und gewinnbringend. Lediglich das Hochladen eines UML-Klassendiagramms schien ihnen Schwierigkeiten bei der Beurteilung zu bereiten. Während des Evaluationsprozesses probierten einige diese Funktionalität auf freiwilliger Basis aus und demonstrierten dies allen weiteren Personen. Dass sie nicht von allen getestet wurde, kann neben der Länge des Fragebogens eine mögliche Erklärung für dieses Ergebnis darstellen.

Auch die Gesamtbewertung fiel positiv aus. Wie in Abbildung 8.3 einsehbar ist, vergaben sie einmal (7,1%) 7, zweimal (14,3%) 8, siebenmal (50,0%) 9 und dreimal (21,4%) 10 Punkte. Von einer Person (7,1%) gab es keine Antwort.

Weiterhin äußerten sie die folgenden Erweiterungswünsche:

- Beim Hochladen habe ich zu spät gemerkt, dass das Lade-Icon rechts oben ist (Hinweis durch den Autor: dies ist durch die Streamlit-Bibliothek so vorgegeben und nicht veränderbar).
- Download mit erhaltenem Feedback in einer Datei.
- Kleiner Button am Ende der Seite, der wieder nach oben führt.
- Anzeige, wie viele Studierende im Chat online sind.

Als weiteres Feedback gab es zwei Anmerkungen:

- Die Chat-Funktion finde ich super wichtig.
- Die Strategiehilfe ist super! Hätte ich sehr gut gebrauchen können.

8.2.5 Evaluation zu dem Programmbereich für die Kollaboration

Der letzte für Studierende vorgesehene Programmbereich ist der für die Generierung kollaborativer Aufgaben.

Auf die geschlossenen Fragen gaben die Teilnehmenden die folgenden Antworten:

- Den strukturellen Aufbau des Programmbereichs Kollaboration bewerte ich als sinnvoll.
  Dieser Äußerung stimmten 21,4% der Studierenden eher und 64,3% voll zu. Zwei Personen (14,3%) konnten dazu keine Aussage treffen.
- Die Erstellung einer eigenen Übungsaufgabe funktioniert problemlos.
  Zu 57,1% stimmten die Studierenden voll zu. Alle weiteren gaben an, dies nicht beurteilen zu können.
- Die automatisierte Erstellung einer Übungsaufgabe funktioniert problemlos.
  Laut 57,1% der Teilnehmenden funktionierte die automatisierte Erstellung einer Übungsaufgabe problemlos. 42,9% gaben an, dies nicht beurteilen zu können.
- Die Auswahlmöglichkeiten bezogen auf das Themengebiet bei der automatisierten Erstellung einer Übungsaufgabe empfinde ich als ausreichend.
  Die Auswahlmöglichkeiten empfanden 7,1% eher nicht, 35,7% eher und 42,9% als ausreichend. Zwei Personen (14,3%) gaben an, dies nicht beurteilen zu können.
- Die Auswahlmöglichkeiten bezogen auf den Schwierigkeitsgrad bei der automatisierten Erstellung einer Übungsaufgabe empfinde ich als ausreichend.

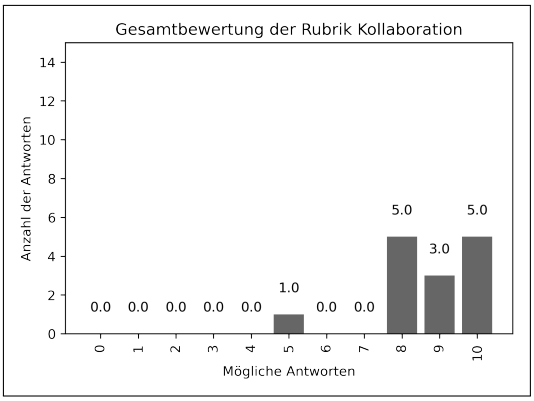


Abbildung 8.4: Darstellung der Gesamtbewertung der Rubrik für die Kollaboration im Rahmen einer Evaluation.

Die Auswahlmöglichkeiten bezogen auf den Schwierigkeitsgrad empfanden 7,1% als eher und 85,7% als vollkommen ausreichend. Lediglich eine Person (7,1%) konnte dies nicht beurteilen.

- Das Hochladen einer Musterlösung hat problemlos funktioniert.
  57,1% stimmten dieser Aussage voll zu. 42,9% konnten dies nicht beurteilen.

- Das automatisierte Erstellen einer Musterlösung hat problemlos funktioniert.
  Unter den Teilnehmenden stimmte eine Person (7,1%) der Aussage eher zu, wohingegen 57,1% voll zustimmten. Alle weiteren (35,7%) konnten dies nicht beurteilen.

Wie durch Abbildung 8.4 nachvollziehbar ist, gab es eine positive Gesamtbewertung der Rubrik. Lediglich eine Person (7,1%) vergab weniger als 8 Punkte. Weitere fünf (35,7%) gaben exakt 8, drei (21,4%) gaben 9 und weitere fünf (35,7%) 10 Punkte.

Die Resultate der geschlossenen Fragen und der Gesamtbewertung zeigen, dass die Studierenden den Programmbereich als durchaus sinnvoll erachten und diesen gut bewerten. Wie auch schon bei der Rubrik für Studierende gab es bei einigen Teilnehmenden Unklarheiten bei den Funktionalitäten für die automatisierten Generierungen.

Als weitere Funktionswünsche nannten die Partizipierenden folgende Punkte:

- Speicherung von eingegebenen Übungsaufgaben. Diese können anschließend von anderen Studierenden zu Übungszwecken abgerufen werden.
- Benutzer Icons, um anzuzeigen, wer sich in einer Session befindet.
- Erstellung der Musterlösung hat gut geklappt. Allerdings wäre ein Tutorial für die Sektion gut, weil man nicht direkt versteht, dass man die XML-Datei als Musterlösung wieder hochladen muss.

Für diese Rubrik gab es keinerlei weiteres Feedback.

8.2.6 Evaluation des erhaltenen Feedbacks

Das erarbeitete Feedback-Modell (siehe Abbildung 6.1) ist in diesem Unterabschnitt zu evaluieren. Eine Überprüfung ist erst durch eine Kombination mit den weiteren Softwarebestandteilen und insbesondere der grafischen Benutzeroberfläche sinnvoll.

Zu den geschlossenen Fragen gaben die Studierenden folgende Antworten:

- Die Rückmeldung zu fehlenden Diagrammkomponenten habe ich problemlos verstanden.
  Die Rückmeldung zu fehlenden Diagrammkomponenten wurde von einer Person (7,1%) nicht, von zwei (14,3%) eher nicht, von drei (21,4%) eher und von sieben (50,0%) sehr gut verstanden. Eine Person (7,1%) konnte dies nicht beurteilen.
- Anhand der Rückmeldung zu fehlenden Diagrammkomponenten kann ich mir erklären, wo mir bei der Modellierung ein Fehler unterlaufen ist.
  Dieser Aussage stimmten die Teilnehmenden zu 7,1% eher nicht, zu 28,6% eher und zu 50,0% voll zu. Zwei Personen (14,3%) gaben an, dies nicht beurteilen zu können.
- Die Rückmeldung zu überflüssigen Diagrammkomponenten habe ich problemlos verstanden.

21,4% der Teilnehmenden gaben an, die Rückmeldung zu überflüssigen Diagrammkomponenten eher nicht problemlos zu verstehen, wohingegen 28,6% eher und 42,9% voll zustimmten. Eine Person (7,1%) konnte dies nicht beurteilen.

- Anhand der Rückmeldung zu überflüssigen Diagrammkomponenten kann ich mir erklären, wo mir bei der Modellierung ein Fehler unterlaufen ist.
  Die Studierenden konnten sich anhand der Rückmeldung zu 7,1% eher nicht, zu 28,6% eher und zu 50,0% sehr gut erklären, wo bei der Modellierung ein Fehler unterlaufen ist. Zwei Personen (14,3%) gaben an, dies nicht beurteilen zu können.

- Die Rückmeldung zu gleichnamigen Diagrammkomponenten mit unterschiedlichen Attributen, Methoden oder Relationen habe ich problemlos verstanden.
  14,3% stimmten dieser Aussage eher nicht, 35,7% eher und 42,9% voll zu. Eine Person (7,1%) konnte dies nicht beurteilen.

- Anhand der Rückmeldung zu gleichnamigen Diagrammkomponenten mit unterschiedlichen Attributen, Methoden oder Relationen kann ich mir erklären, wo mir bei der Modellierung ein Fehler unterlaufen ist.
  Die Studierenden konnten zu 7,1% eher nicht und zu jeweils 42,9% eher und sehr gut nachvollziehen, wo ihnen ein Fehler unterlaufen ist. Eine Person (7,1%) gab an, dies nicht beurteilen zu können.

- Die Rückmeldung der zu erkennenden Elemente in der textuellen Anforderungsbeschreibung empfinde ich als hilfreich.
  Eine Person (7,1%) gab an, dies eher nicht beurteilen zu können. Weiterhin stimmten 21,4% eher und 64,3% voll zu, während eine Person (7,1%) keine Beurteilung abgeben konnte.

- Die Tipps zu den fehlenden Klassen innerhalb des studentischen UML-Klassendiagramms empfinde ich als hilfreich.
  Die Tipps wurden zu 7,1% als eher nicht, zu 21,4% als eher und zu 64,3% als sehr hilfreich empfunden. Eine Person (7,1%) konnte dies nicht beurteilen.

- Die Tipps zu den überflüssigen Klassen innerhalb des studentischen UML-Klassendiagramms empfinde ich als hilfreich.
  Die Tipps empfanden 7,1% als eher nicht, 7,1% als eher und 78,6% als sehr hilfreich. Eine Person (7,1%) konnte dies nicht beurteilen.

- Die Tipps zu den Attributen empfinde ich als hilfreich.
  Jeweils 7,1% der Teilnehmenden stimmten dieser Aussage eher nicht sowie eher zu. Dagegen gaben 78,6% ihre volle Zustimmung während eine Person (7,1%) keine Beurteilung abgeben konnte.

- Die Tipps zu den Methoden empfinde ich als hilfreich.
  Die Tipps sind von 7,1% als eher nicht, von 14,3% als eher und von 71,4% als sehr hilfreich eingestuft worden. Eine Person (7,1%) konnte dies nicht beurteilen.

- Die Tipps zu den Beziehungen der Diagrammkomponenten untereinander empfinde ich als hilfreich.
  Je 7,1% empfanden die Tipps als eher nicht oder eher hilfreich. 78,6% bewerteten sie als sehr hilfreich. Eine Person (7,1%) konnte dies nicht beurteilen.

- Die Tipps zu den zu wiederholenden Kursinhalten empfinde ich als hilfreich.
  Drei der Studierenden (21,4%) stimmten eher und zehn (71,4%) voll zu. Eine Person (7,1%) konnte dies nicht beurteilen.

- Die Empfehlung zum weiteren Vorgehen empfinde ich als hilfreich.
  Die Empfehlung für das weitere Vorgehen wurde von 35,7% als eher und von 57,1% als sehr hilfreich empfunden. Eine Person (7,1%) konnte dies nicht beurteilen.

Auch die Evaluation des Feedbacks anhand der geschlossenen Fragen fällt überwiegend sehr positiv aus. Die Studierenden können alle angezeigten Inhalte nachvollziehen und empfinden die bereitgestellten Tipps als sehr hilfreich. Dies spiegelt sich auch in der in Abbildung 8.5 dargestellten Gesamtbewertung wider, bei der 71,4% aller Teilnehmenden mindestens 8 Punkte vergeben haben. Konkret gab es eine Bewertung (7,1%) mit 4, drei (21,4%)

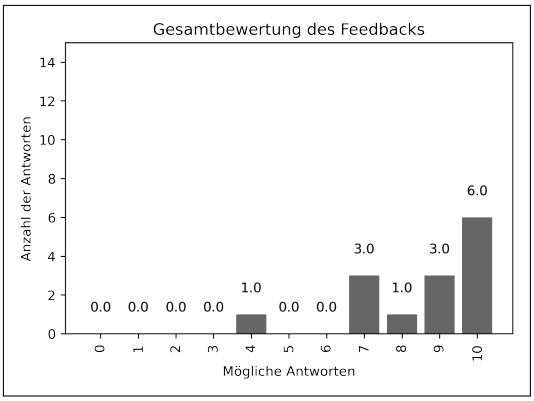


Abbildung 8.5: Darstellung der Gesamtbewertung für das Feedback im Rahmen einer Evaluation.

mit 7, eine (7,1%) mit 8, drei (21,4%) mit 9 und sechs (42,9%) mit 10 Gesamtpunkten.

Für potenzielle Funktionserweiterungen wurden die folgenden Punkte gewünscht:

- Art der Beziehungen beim Feedback besser kenntlich machen.
- XML-Datei, in der man seine Fehler markiert bekommt (Hinweis durch den Autor: diese wäre anschließend nicht mehr in einem Modellierungseditor einlesbar. Zudem würde es sich um eine schwer zu überblickende Darstellung, in Form von XML, handeln.).
- Markierungen im Skript für Stoff, der wiederholt werden soll.
- Beschreibende Sätze für fehlende Klassen wären besser.
- Farbliche Hervorhebungen von Klassen-, Attribut- und Methodennamen.
- Hinweise auf Seiten im Skript anhand des Fehlerprofils.

Weiteres Feedback für diese Rubrik gab es nicht.

8.2.7 Allgemeine Evaluation des Systems

Zum Abschluss der Evaluation gab es für die Teilnehmenden die Möglichkeit, eine allgemeine Rückmeldung zu dem System zu geben.

Die folgenden Antworten gab es auf die geschlossenen Fragen:

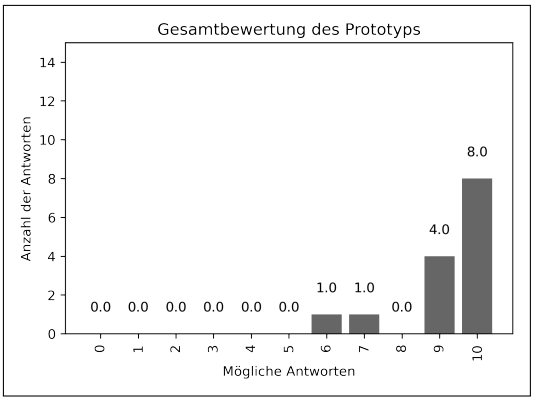


Abbildung 8.6: Darstellung der Gesamtbewertung einer allgemeinen Evaluation des Prototyps.

- Ich würde den Prototyp verwenden, um mein Wissen zu UML-Klassendiagrammen auf- oder auszubauen.
  Dieser Aussage stimmten 7,1% eher und 92,9% voll zu.

- Ich würde den Prototyp verwenden, um mich auf Prüfungsaufgaben zu UML-Klassendiagrammen vorzubereiten.
  7,1% würden den Prototyp eher und 92,9% definitiv für die Prüfungsvorbereitung nutzen.

- Meinen Kommiliton*innen würde ich die Nutzung des Prototyps empfehlen.
  92,9% stimmen dieser Aussage voll zu. Eine Person (7,1%) konnte dies nicht beurteilen.

Die allgemeine Evaluation anhand der geschlossenen Fragen fiel besonders positiv aus. Fast alle Teilnehmenden würden den Prototyp für die Unterstützung ihres Lernprozesses einsetzen und ihn darüber hinaus auch noch den Mitstudierenden empfehlen. Die in Abbildung 8.6 visualisierte Gesamtbewertung untermauert dieses sehr gute Ergebnis. Lediglich zwei Personen (je 7,1%) werteten weniger als 9 Punkte. Die weiteren 85,7% der Teilnehmenden vergaben 9 (viermal mit insgesamt 28,6%) oder 10 (achtmal mit insgesamt 57,1%) Punkte.

Als weiterführende Wünsche bezüglich der Funktionalität führten die Studierenden folgende Punkte an:

- Bereich, in dem man seinen Lernfortschritt verfolgen kann.

- Anhand des Fehlerprofils erkennen, wo man selbst am meisten Fehler macht.
- Grafische Darstellung der Musterlösung.
- Größe des UML-Editors sollte beliebig wählbar sein (Hinweis durch den Autor: aktuell steht nur die standardisierte Anzeige sowie ein Vollbildmodus auf die gesamte Breite der grafischen Benutzeroberfläche zur Verfügung).
- Lerntipps durch Künstliche Intelligenz im Tutorial Bereich anzeigen.

Als weiteres Feedback nannten die Teilnehmenden folgendes:

- Super Idee und Umsetzung.
- Sehr gerne in der Lehre einsetzen.
- Kollaborationsmodus sehr sinnvoll.
- Unbegrenzte Anzahl an Aufgaben super.
- Chat-Funktion super.
- Tool berücksichtigt alle wichtigen Aspekte.
- Sehr hilfreich.

8.3 Zusammenfassung der Ergebnisse

Die in dem vorherigen Abschnitt angeführten Ergebnisse der Evaluation verdeutlichen, dass das konzipierte und prototypisch implementierte System von Studierenden als sehr vielversprechend zu beurteilen ist. Durch eine ansprechende und einfach zu bedienende Benutzeroberfläche finden sie sich schnell zurecht. Auch die einzelnen, für diese Zielgruppe zugänglichen Rubriken fanden Anklang. Das nach einem Vergleich zweier UML-Klassendiagramme erhaltene Feedback war für die meisten Teilnehmenden gut verständlich und wurde darüber hinaus als hilfreich wahrgenommen. Die finale Gesamtbewertung im speziellen verdeutlicht, dass die Studierenden das System äußerst positiv wahrnehmen und diesen für den Einsatz in der hochschulischen Lehre als durchaus adäquat erachten. Ein solches kumuliertes

Ergebnis für das Artefakt ist im Hinblick auf die Fragestellungen dieser Arbeit von entscheidender Bedeutung. Erklärbar ist es beispielsweise anhand der konsequenten Verfolgung des DSR-Forschungsdesigns nach Hevner u. a. (2004). Auf Basis dieser Vorgehensweise erfolgte eine Untersuchung der Wissensbasis sowie eine Erhebung weiterer Anforderungen anhand der Umgebung. Die positiven Resultate spiegeln diese umfassenden Analysen wider und verdeutlichen ebenfalls, dass entsprechende Funktionalitäten durch den Prototyp anschaulich sowie effektiv zur Verfügung gestellt werden. Die Forschungsfragen der Arbeit lassen sich demnach nicht nur beantworten, sondern auch zufriedenstellend beantworten.

KAPITEL 9

Zusammenfassung und Ausblick

Das letzte Kapitel der Arbeit hebt den Innovationscharakter der vorgestellten Ergebnisse hervor. Weiterhin erfolgt eine Zusammenfassung der zentralen Punkte, bevor einige zukünftige Weiterentwicklungsmöglichkeiten der Ergebnisse beschrieben sind.

9.1 Beschreibung des Innovationscharakters

Das vorgestellte System, bestehend aus einem Client (Frontend) respektive einem Server (Backend), weist gegenüber bestehenden Softwarelösungen deutlichen Innovationscharakter auf, wenn es um die computergestützte, hochschulische Lehre von UML-Klassendiagrammen geht.

Eine Untersuchung verwandter Arbeiten ist in den Kapiteln 3, 4, 5 sowie 6 präsentiert worden. Bei den Entwicklungen handelt es sich um Neubeziehungsweise Weiterentwicklungen der in der Wissensbasis bestehenden Ansätze.

Als erste zu nennende Innovation gilt die automatisierte Generierung textueller Anforderungsbeschreibungen mit zugehörigen Aufgabenstellungen in deutscher Sprache. Diese sind mit unterschiedlichen Schwierigkeitsgraden sowie akademisch orientierten Themenschwerpunkten erstellbar und können zusätzlich Beschreibungen von Entwurfsmustern enthalten. Auch die

Inklusion einer solchen Komponente in ein Softwaresystem, mit dem sich sowohl Dozierende als auch Studierende für sich selbst oder in Kollaboration Übungsaufgaben erzeugen können, ist ein Novum. Sie unterstützt dabei, das digitale Lehren und Lernen von UML-Klassendiagrammen zu ermöglichen.

Weiterhin zeigt die Analyse der Wissensbasis, dass es bis dato keinen Ansatz gab, um in einem deutschsprachigen, akademischen Rahmen textuelle Anforderungsbeschreibungen automatisiert in eine Musterlösung im XML-Format zu überführen. Aufgrund linguistischer Differenzen (beispielsweise im Satzbau) ist eine simple Adaption bestehender Lösungen nicht möglich und erforderte daher eine Neuentwicklung für den angeführten Problemkontext.

Aufgrund ihrer gemeinsamen Entwicklung, wie in Kapitel 4 einsehbar ist, sind die beiden Softwarekomponenten aufeinander abgestimmt und erzielen gute Ergebnisse. Eine weitere Innovationscharakteristik ergibt sich aus dieser Symbiose. So ist anhand der Literatur kein Ansatz identifizierbar, der in Bezug auf die Hochschullehre im deutschsprachigen Raum Übungsaufgaben mit zugehörigen Musterlösungen generiert, die weiterhin auch noch eine Modellierung von Entwurfsmustern erfordern können.

Bei einem Vergleich zweier UML-Klassendiagramme miteinander sind häufig fest definierte Modellierungseditoren vorgeschrieben. Die hierfür in Kapitel 5 entwickelte Lösung ist hiervon unabhängig, da sowohl XML- als auch Bilddateien einlesbar sind. Weiterhin fokussieren sich die in der Wissensbasis enthaltenen Ansätze nicht auf das Auslesen deutschsprachiger Namen für Diagrammbestandteile.

Eine Konzipierung der Softwarekomponente für die Generierung von Feedback findet sich in Kapitel 6. Sie stellt eine Kombination sowie Erweiterung verwandter Arbeiten dar und lässt sich daher von diesen abgrenzen.

Die SLR von Huber und Hagel (2022b) verdeutlicht neben den weiteren Resultaten aus Kapitel 3, dass in der Literatur präsentierte Softwareanwendungen für die computergestützte Lehre von UML-Klassendiagrammen bis dato keine ganzheitliche, digitale Abbildung des Übungsbetriebs ermöglichen. Häufig adressieren sie individuelle Herausforderungen, die Lehrpersonen bei ihren Studierenden identifizieren konnten. Eine Zusammenführung der erläuterten Komponenten stellt daher eine zentrale Neuerung dar, die durch diese Arbeit realisierbar ist. Gemeinsam stellen sie grundlegen-

de Kernfunktionalitäten bereit, um einen ganzheitlichen, rechnergestützten Ansatz zu ermöglichen, der den gesamten Lernprozess von Studierenden vom Erhalt einer Übungsaufgabe bis hin zum Empfang individuellen Feedbacks begleitet. Entgegen einer Fixierung auf Beobachtungen durch Dozierende fanden Anforderungen bestehender Tools sowie die Wünsche aller Hauptakteure Einzug in die Konzeptionierung. Nur so ist ein vollumfängliches System planbar, das keine Insellösung für Individualherausforderungen darstellt. Diese Ganzheitlichkeit ist als zentrale Innovation des Systems anzuführen. Sie ermöglicht es Dozierenden, Übungsaufgaben und weitere relevante Inhalte für Studierende im Rahmen einer Übungseinheit zur Verfügung zu stellen. Textuelle Anforderungsbeschreibungen sowie Musterlösungen sind dabei entweder selbst oder automatisiert erstellbar. Letzteres kann Lehrpersonen merklich entlasten. Zusätzlich können sich Studierende benötigtes Wissen in einer eigens dafür angelegten Rubrik aneignen und durch simple Übungsaufgaben vertiefen. Verwandte Arbeiten setzen häufig die Anwesenheit von Dozierenden voraus, um beispielsweise Inhalte bereitzustellen. Das vorgestellte System setzt dies nicht voraus. Sofern sich die Studierenden eine automatisierte Übungsaufgabe erzeugen, können sie eigenständig und ohne zeitliche Einschränkungen üben. Auch die Möglichkeit der kollaborativen Zusammenarbeit stellt ein Novum dar. Die Rubrik ermöglicht die Eingabe eigener Anforderungsbeschreibungen und Musterlösungen, kann diese jedoch auch maschinell generieren. Wie bereits erwähnt, ist es Lernenden meist nicht möglich selbst zu verifizieren, ob und zu welchem Grad es sich bei ihrem Klassendiagramm um eine valide Lösung handelt. Durch seinen Funktionsumfang wirkt das System auch dieser Herausforderung entgegen.

Aus den angeführten Gründen ist argumentierbar, dass es sich bei der Konzeptionierung um einen innovativen Ansatz handelt, der sich anhand diverser Charakteristiken von bestehenden Arbeiten abgrenzen lässt.

9.2 Zusammenfassung der wesentlichen Ergebnisse dieser Arbeit

Die folgenden Unterabschnitte sollen die vorliegende Arbeit zusammenfassen und die Quintessenz der einzelnen Kapitel präsentieren.

9.2.1 Problemkontext und Forschungsdesiderat

In der hochschulischen Lehre von Software Engineering und im speziellen von UML-Klassendiagrammen stehen Studierende in der Lehrpraxis vor wiederkehrenden Problemen, die auch in der Literatur vielfach dokumentiert sind. Zu nennen ist beispielsweise die Extraktion beschriebener Softwarekomponenten aus natürlichsprachlichen Texten. Anschließend sind diese zu abstrahieren und in ein Diagramm zu überführen. In vielen anderen Disziplinen ist es häufig möglich, ein exaktes Ergebnis respektive einen Lösungsraum zu bestimmen. Bei Modellierungsaufgaben in diesem Kontext ist dies jedoch nicht umsetzbar. Die Resultate können sich visuell und inhaltlich von einer Musterlösung unterscheiden und dennoch korrekt sein. Aus diesem Grund erhalten Lernende meist auch kein individuelles und zeitnahes Feedback. Ihre Lösungen müssten zuvor einzeln auf mögliche Fehler inspiziert werden. Für Dozierende ist dies bereits bei kleineren Übungsgruppen aus zeitlichen Gründen nicht mehr realisierbar. Üblicherweise erhalten sie daher eine Musterlösung, anhand derer sie Rückschlüsse auf ihre eigenen Diagramme ziehen sollen. Dieses Vorgehen ist jedoch suboptimal, da Studierende gerade zu Beginn ihres Lernprozesses nur selten verifizieren können, ob eine abweichende Modellierung ebenfalls zu einem korrekten Ergebnis geführt hätte. Ihr Erfolg könnte durch individualisierte Hilfestellungen, Tipps und weitere Formen der Rückmeldung deutlich gesteigert werden.

Eine zusätzliche Herausforderung in der Lehrpraxis ist die eingeschränkte Verfügbarkeit von Übungsbeispielen. Die Erstellung dieser kann sich für Lehrpersonen zeitintensiv gestalten, wenn Aufgaben mit unterschiedlichen Schwierigkeitsgraden und an den Studiengängen orientierten Themenschwerpunkten zur Verfügung zu stellen sind. Weiterhin benötigen die Studierenden auch eine Form von Rückmeldung, selbst wenn es sich dabei lediglich um eine Musterlösung handelt.

In der bestehenden Literatur gibt es Softwareprogramme, mit denen diesen und weiteren Herausforderungen entgegengewirkt werden soll. Oftmals sollen diese jedoch nur spezifischen Problemstellungen entgegenwirken und stellen keine ganzheitliche Lösung dar. Daraus ist die zentrale Forschungsfrage dieser Arbeit abzuleiten:

> **Wie kann ein computergestütztes System zur Lehre von UML-Klassendiagrammen für die Unterstützung der hochschulischen Lehre von Software Engineering konzipiert und prototypisch implementiert werden?**

Aus dieser ergeben sich drei Unterfragen sowie diverse wissenschaftliche Zielsetzungen, die in Kapitel 1 nachzulesen sind.

Da die Planung und prototypische Entwicklung eines Softwaresystems Hauptbestandteil des Vorhabens sind, eignet sich das DSR-Forschungsdesign nach Hevner u. a. (2004) für die Beantwortung der Forschungsfragen und für die Erreichung der wissenschaftlichen Zielsetzung.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- In der hochschulischen Lehre von Software Engineering und im speziellen bei UML-Klassendiagrammen gibt es für Studierende aber auch Dozierende eine Vielzahl von Herausforderungen.
- Unterstützende Softwaresysteme decken meist nur einzelne Herausforderungen ab.
- Bisher gibt es keinen ganzheitlichen Ansatz zur Lösung der Problemstellung.
- Es lässt sich eine zentrale Forschungsfrage mit drei Unterfragen und diversen wissenschaftlichen Zielsetzungen ableiten.
- Zur Beantwortung respektive Erfüllung dieser eignet sich das DSR-Forschungsdesign nach Hevner u. a. (2004).

9.2.2 Theoretische Betrachtung grundlegender und kontextuell relevanter Aspekte

Software Engineering gilt als eine vergleichbar junge Disziplin, deren Fokus auf der systematischen Planung, Entwicklung, Inbetriebnahme sowie der Wartung von Software liegt. Dessen Tätigkeitsfelder sind durch Normen sowie Grundlagenwerke beschrieben.

Gerade durch die zunehmende Digitalisierung und Vernetzung wird es immer relevanter, robuste und erweiterbare Softwaresysteme bereitzustellen. In der Hochschullehre sollen Studierende informatiknaher Studiengänge mit diesen Konzepten vertraut gemacht und auf zukünftige berufliche Herausforderungen vorbereitet werden. Jedoch stellt gerade die Überführung von Anforderungen in ein Softwaredesign in Form von UML-Diagrammen die Studierenden vor Probleme. Diese Modellierungssprache entstand als Vereinheitlichung vieler vorangegangener Ansätze, mit denen versucht wurde, die zunehmende Komplexität moderner Softwaresysteme handhabbar zu machen.

Gerade in den vergangenen Jahren haben Deep-Learning-Modelle in unterschiedlichsten Bereichen, wie beispielsweise der Objekterkennung oder Verarbeitung natürlicher Sprache, beeindruckende Erfolge erzielt. Der Einsatz solcher Technologien konnte im Kontext der Arbeit zur Lösung des erläuterten Problemkontexts führen.

Das avisierte Resultat soll ein studierendenzentriertes Lehren und Lernen ermöglichen. Wie der Name bereits verlauten lässt, stehen Studierende und nicht Dozierende hierbei im Fokus der Bemühungen. Letztere nehmen dabei eine begleitende und unterstützende Rolle bei der Wissensaneignung ein. Da solche Gestaltungsformen die intrinsische Motivation von Lernenden erhöhen können, erfreuen sie sich zunehmender Beliebtheit. Von zentraler Bedeutung ist hier das Feedback, welche eine Rückmeldung zu einer vorangegangenen Tätigkeit darstellt und eine Einschätzung über deren Güte ermöglicht. Für die Kommunikation von Feedback gibt es in der hochschulischen Lehre diverse Vorgehensweisen. Es lässt sich weiterhin in unterschiedliche Typen separieren und ist, wie am Beispiel von Hattie und Timperley (2007) verdeutlicht, in ganze Verfahrensmodelle integrierbar.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Der Fokus der Fachdisziplin Software Engineering liegt auf der systematischen Planung, Entwicklung, Inbetriebnahme sowie der Wartung von Software.
- Gerade in der Hochschullehre informatiknaher Studiengänge stellt die Lehre dieser einen zentralen Aspekt dar.
- Deep-Learning-Modelle konnten in den vergangenen Jahren in unterschiedlichsten Bereichen große Erfolge erzielen und können bei der Lösung der erläuterten Problemstellung unterstützen.
- Das Ergebnis soll ein studierendenzentriertes Lehren und Lernen ermöglichen.
- Feedback kann auf verschiedene Weisen kommuniziert werden und ist je nach Art in diverse Typen gruppierbar.
- Es gibt Verfahrensmodelle für effektives Feedback, von denen eines vorgestellt wurde.

9.2.3 Untersuchung und Erweiterung der bestehenden Wissensbasis

Das DSR-Forschungsdesign nach Hevner u. a. (2004) sieht neben der Planung und Entwicklung von Theorien sowie Artefakten auch eine Untersuchung der bestehenden Wissensbasis vor. Verwandte Arbeiten dienen dabei als Ausgangsbasis und sind vom Vorhaben der Arbeit abzugrenzen. Bei einem SLR handelt es sich um ein anerkanntes, reglementiertes und strukturiertes Vorgehen für die Identifikation und Zusammenführung von wissenschaftlichen Beiträgen. Im Kontext der Arbeit standen dabei drei Fragen im Mittelpunkt. Nach der Definition von Inklusions- und Exklusionskriterien, der Festlegung zu durchsuchender, elektronischer Datenbanken sowie eines komplexen Suchbegriffs erfolgte die Durchführung. Alle relevanten Veröffentlichungen wurden gelesen und auf unterschiedliche Bestandteile untersucht.

Die Ergebnisse untermauern die im Problemkontext getroffene Aussage, dass keine vergleichbare Softwarelösung einen ganzheitlichen Ansatz abbilden kann. Die gewonnenen Erkenntnisse sind jedoch nicht hinreichend, um alle Fragen nach benötigten Funktionalitäten abzudecken. Es ist erkennbar, dass Lehrpersonen und Forschende die Anwendungen häufig auf eigenen Beobachtungen sowie Annahmen stützen. Eine umfassende Befragung von Studierenden, welche die Hauptzielgruppe solcher Unternehmungen darstellen, kam bis dato zu kurz. Um ein vollständigeres Bild erhalten zu können, ist auch eine Befragung von Dozierenden zu ihren Wünschen wichtig, wenn eine Übungseinheit zu UML-Klassendiagrammen digital abzubilden ist. Daher wurden diese beiden Gruppen im Rahmen von qualitativen, leitfadengestützten Experteninterviews befragt.

Aus den Ergebnissen des SLR sowie der Interviews ließ sich ein umfassender Anforderungskatalog definieren, der als Grundlage für alle weiteren Entwicklungen der Arbeit herangezogen wurde.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Dem DSR-Forschungsdesign entsprechend ist eine Untersuchung der Wissensbasis im Rahmen eines SLR erfolgt.
- Die Ergebnisse untermauern im Problemkontext beschriebene Herausforderungen und sind der Ausgangspunkt für weitere Entwicklungen.
- Eine Abgrenzung zu bestehenden Ansätzen muss vonstattengehen.
- Daraus gewonnene Erkenntnisse sind im Rahmen der Arbeit nicht hinreichend. Es muss daher eine Befragung der Zielgruppen stattfinden.
- Alle Resultate des Kapitels wurden in einem Anforderungskatalog festgehalten.
- Der Anforderungskatalog ist der Ausgangspunkt für alle weiteren geplanten Schritte.

9.2.4 Konzipierung und prototypische Implementierung von Softwarekomponenten für die Erzeugung textueller Übungsaufgaben sowie Musterlösungen

Die Erstellung zentraler Artefakte für die Bereitstellung der benötigten Funktionalitäten erfolgte in drei unterschiedlichen Softwarekomponenten. Da diese auch individuell voneinander in andere Softwareumgebungen integrierbar sein sollen, war eine zusätzliche Untersuchung in der Wissensbasis vorhandener, verwandter Arbeiten vorgesehen, um dem zyklischen Vorgehen des Forschungsdesigns zu entsprechen. Anhand derer ist erkennbar, dass es keine softwaregestützte Lösung für die Generierung von Übungsaufgaben für die deutschsprachige Hochschullehre von UML-Klassendiagrammen gibt, die den in Kapitel 4 genannten Kriterien entsprechen. Auch eine Überführung deutschsprachiger, textueller Anforderungsbeschreibungen in Klassendiagramme ist nicht ohne Weiteres möglich.

In einem ersten Schritt befasste sich das Kapitel mit der Konzeption und anschließend der prototypischen Implementierung der Komponente für die Erstellung von Übungsaufgaben. Das Softwaredesign folgt dem Fassade-Entwurfsmuster. Mit den Bestandteilen des Pakets kann über eine zentrale Schnittstelle kommuniziert werden. Außerdem orchestriert die sogenannte Fassadeklasse den iterativen Aufruf einzelner Klassen und deren Methoden, um die gewünschten Ergebnisse bereitzustellen. In Anlehnung an mehrere Prinzipien des Softwaredesigns sowie der Softwarearchitektur sind alle Bestandteile logisch voneinander getrennt und erhalten nur eine individuelle Aufgabe. Das Gleiche gilt für alle folgenden Softwarekomponenten der Arbeit. Die generierten, textuellen Anforderungsbeschreibungen verwenden fest definierte Signalworte und sind durch die Trainingsdaten auf einen vorgegebenen, strukturellen Aufbau getrimmt.

Ohne diese Grundvoraussetzung ist eine Umwandlung natürlichsprachlichen Texts in ein UML-Klassendiagramm in Form einer XML-Datei nicht möglich. Die deutsche Sprache ist zu komplex, als dass ein allgemeingültiges Vorgehen identifizierbar ist. Mit dieser Struktur als Ausgangsbasis ist in Kapitel 4 ein umfassendes Regelwerk definiert, anhand dessen diese Überführung möglich ist. Auch hier folgt das Softwaredesign dem Fassade-Entwurfsmuster.

Folglich sind beide Bestandteile über ihre jeweiligen Fassadeklassen ansprechbar und gemeinsam in ein Gesamtsystem integrierbar. Der modulare Aufbau ermöglicht jedoch auch eine individuelle sowie unabhängige Nutzung. Durchgeführte Evaluationen bestätigen die korrekte Funktionsweise dieser. Mit Abschluss dieses Kapitels sind sowohl eine der Unterforschungsfragen als auch wissenschaftliche Zielsetzungen beantwortbar respektive erfüllbar.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Verwandte Arbeiten werden pro konzipiertem Softwareartefakt untersucht.
- Bisher gab es keine Lösung für die Generierung deutschsprachiger textueller Anforderungsbeschreibung, die den angeführten Kriterien entsprechen.
- Die resultierenden Übungsaufgaben beinhalten vordefinierte Signalworte und Strukturen, um eine automatisierte Überführung in UML-Klassendiagramme zu ermöglichen.
- Das Kapitel konzipiert und implementiert (prototypisch) beide Softwarekomponenten.
- Mit einem umfassenden Regelwerk ist eine solche Überführung möglich.
- Beide Softwarekomponenten besitzen einen modularen Aufbau und folgen dem Fassade-Entwurfsmuster.
- Das Kapitel beantwortet eine Unterforschungsfrage und erfüllt beschriebene, wissenschaftliche Zielsetzungen.

9.2.5 Konzipierung und prototypische Implementierung einer Softwarekomponente für den Vergleich zweier UML-Klassendiagramme

Damit Studierende eine Rückmeldung zu ihren UML-Klassendiagrammen erhalten können, sind diese mit einer Musterlösung zu vergleichen. Detek-

tierte Differenzen sollen in einem Folgeschritt in Form von effektivem Feedback an die Studierenden kommuniziert werden.

Eine Untersuchung der Wissensbasis verdeutlicht, dass bestehende Publikationen sich von der entwickelten Softwarekomponente in einigen wesentlichen Punkten unterschiedet. Die Konzeption berücksichtigt sowohl mit einer Modellierungsanwendung erstellte XML-Dateien als auch Bilddateien von Klassendiagrammen als Eingabe. Dabei können sowohl Lehrende als auch Lernende beliebig zwischen diesen Formaten wählen. In ersterem Fall ist die XML-Struktur direkt in eine interne Repräsentation überführbar. Liegt eine Bilddatei vor, müssen in einem Zwischenschritt vorerst alle abgebildeten Diagrammbestandteile detektiert, lokalisiert und klassifiziert werden, bevor eine Zuordnung von Beziehungen der Komponenten sowie eine Erkennung enthaltener Texte durchführbar ist. Stehen all diese Informationen zur Verfügung, sind diese ebenfalls in einer internen Repräsentation speicherbar. Sie ermöglichen einen Vergleich und die Detektion aller Differenzen.

Auch hier kam als vorherrschendes Softwaredesign das Fassade-Entwurfsmuster zum Einsatz. Über eine Fassadeklasse als Kommunikationsschnittstelle ist das Paket individuell ansprechbar, aber auch in eine Softwareumgebung integrierbar. Durch den modularen Aufbau sind Änderungen respektive Erweiterungen problemlos möglich. Eine Evaluation des Resultats ergab, dass alle relevanten UML-Klassendiagrammkomponenten miteinander verglichen und alle Differenzen erkannt werden können. Mit Abschluss des Kapitels 5 sind sowohl eine der Unterforschungsfragen als auch wissenschaftliche Zielsetzungen beantwortbar und erfüllbar.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Ein Vergleich zweier UML-Klassendiagramme ist notwendig, um Studierenden eine Rückmeldung zu ihren Modellierungen bieten zu können.
- Die Wissensbasis kann die gestellte Unterforschungsfrage nicht hinreichend beantworten.
- Die Softwarekomponente muss sowohl UML-Klassendiagramme als Bilddateien und in Form von XML-Dateien entgegennehmen und in

eine interne Repräsentation überführen.

- Liegt eine Bilddatei als Eingabe vor, sind darin enthaltene Komponenten zu detektieren, lokalisieren und klassifizieren. Anschließend müssen Beziehungen zugewiesen und Texte ausgelesen werden.
- Das Softwaredesign folgt dem Fassade-Entwurfsmuster und ist individuell nutzbar oder in eine Softwareumgebung integrierbar.
- Eine Evaluation zeigt, dass alle kontextuell relevanten Differenzen zweier UML-Klassendiagramme erkennbar sind.
- Das Kapitel beantwortet eine Unterforschungsfrage und erfüllt beschriebene wissenschaftliche Zielsetzungen.

9.2.6 Konzipierung und prototypische Implementierung einer Softwarekomponente für die Bereitstellung von Feedback

Die letzte zentral vorgestellte Softwarekomponente in Kapitel 6 ist für die Bereitstellung von effektivem Feedback verantwortlich. Die bestehende Wissensbasis zeigt auf, dass es diverse Möglichkeiten für eine computergestützte Rückmeldung an Studierende für diese Art der Aufgabe gibt. Allerdings stellen nur sehr wenige Feedback vom Typ „Informatives Lehren“ bereit. Die Konzeption nimmt diese verwandten Arbeiten als Ausgangspunkt und kombiniert beziehungsweise erweitert diese. Dafür wurde ein eigenes Verfahrensmodell in Anlehnung an Hattie und Timperley (2007) entwickelt.

Zuvor detektierte Differenzen zweier UML-Klassendiagramme stellen die Grundlage für alle Verarbeitungsschritte dar. Die Konzeptionierung verfolgt zwar auch hier einen modularen Aufbau mithilfe des Fassade-Entwurfsmusters, jedoch ist eine Verwendung der Softwarekomponenten nur in Symbiose mit weiteren sinnvoll.

Die generierte Rückmeldung besteht aus unterschiedlichen Bausteinen, die in Summe zur Verbesserung des Lernerfolgs von Studierenden beitragen können. Die Effektivität des Verfahrensmodells ist durch die finale Evaluation zu überprüfen.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Die Wissensbasis verdeutlicht, dass es diverse Möglichkeiten für Feedback in diesem Kontext gibt.
- Bestehende Ansätze erreichen nur selten eine Rückmeldung des Typs „Informatives Lehren“ und sind selbst dann noch erweiterbar.
- Verwandte Arbeiten dienen als Ausgangsbasis für die Entwicklung eines eigenen Feedback Modells.
- Dieses stellt unterschiedliche Bausteine bereit, die in Kombination zu einer Verbesserung des Lernerfolgs bei Studierenden beitragen sollen.
- Detektierte Differenzen zweier UML-Klassendiagramme stellen eine Grundlage für alle weiteren Verarbeitungsschritte dar. Die Softwarekomponente ist sinnvollerweise daher in Symbiose mit weiteren einzusetzen.
- Da die Softwarekomponente nicht individuell anwendbar ist, erfolgt eine Evaluation mit der Gesamtevaluation des Systems.
- Das Kapitel beantwortet eine Unterforschungsfrage und erfüllt beschriebene wissenschaftliche Zielsetzungen.

9.2.7 Konzipierung und prototypische Implementierung eines Gesamtsystems

Die zuvor präsentierten Softwarekomponenten wurden in Kapitel 7 zu einer Backend-Struktur zusammengeführt. Diese ist für die Bereitstellung von Funktionalität zuständig und folglich von der reinen Anzeige dieser Inhalte abzugrenzen, welche in einem Frontend vorzunehmen ist. Beide Bausteine sind daher theoretisch auch getrennt voneinander nutzbar.

Nicht alle der in Kapitel 3 erhobenen Anforderungen sollten durch die Konzeption berücksichtigt werden. So wurden einzelne unter Nennung eines triftigen Grunds exkludiert. Alle Weiteren sind durch die Softwaredesigns abgedeckt.

Das Backend setzt sich aus unterschiedlichen Softwarepaketen zusammen, unter denen sich auch die zuvor angeführten Softwarekomponenten finden. Zudem implementiert es einen Server, mit dem über das HTTP-Protokoll und seinen entsprechenden Operationen kommuniziert werden kann. Auch das Frontend folgt einem modularen Aufbau und stellt vier Rubriken für die Akteure des Systems bereit. Um die Benutzerschnittstelle ansprechend und intuitiv zu gestalten, wurden einige Designentscheidungen getroffen und erläutert.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Unterteilung des Systems in Back- und Frontend.
- Einzelne Anforderungen wurden aus triftigem Grund von der Umsetzung exkludiert.
- Beide Systembestandteile sind modular aufgebaut und flexibel erweiterbar.
- Das Kapitel beantwortet die zugrundeliegende Forschungsfrage und erfüllt beschriebene wissenschaftliche Zielsetzungen.

9.2.8 Evaluation des Prototyps anhand der Hauptzielgruppe

Die Evaluation des Prototyps erfolgte in Form einer quantitativen Untersuchung mithilfe eines Fragebogens. Mit diesem wurden die Hauptakteure des Systems, die Studierenden, zu unterschiedlichen Themenschwerpunkten befragt. Neben einer Gesamtbewertung für diese gab es geschlossene und offene Fragen. Die Gruppe der Teilnehmenden setzte sich aus Studierenden unterschiedlichen Alters und Geschlechts zusammen, die außerdem diverse Studiengänge belegen.

Die Resultate fallen sehr positiv aus und zeigen, dass die konzipierte sowie prototypische implementierte Anwendung für die Unterstützung der hochschulischen Lehre von UML-Klassendiagrammen gut geeignet ist. Aufgrund der umfassenden Anforderungsanalyse mit allen wichtigen Akteuren ist es möglich, die im Problemkontext erläuterten Herausforderungen anzugehen.

Stichpunktartige Zusammenfassung der zentralen Ergebnisse

- Die Evaluation erfolgte in Form einer quantitativen Untersuchung mithilfe eines Fragebogens.
- Mit diesem wurden die Studierenden als Hauptakteure des Systems befragt.
- Der Fragebogen ist in unterschiedliche Themenschwerpunkte unterteilt.
- Die Resultate fallen sehr positiv aus und zeigen die Eignung des Systems für die Hochschullehre.
- Die im Problemkontext erläuterten Herausforderungen sind durch die Anwendung überwindbar.

9.3 Mögliche Weiterentwicklungen der Ergebnisse

Die Konzeption sieht die Umsetzung der in Kapitel 3 erhobenen Anforderungen vor. Dabei nicht berücksichtigt sind jedoch Bausteine, die häufig in Softwaresystemen enthalten sind. So ist beispielsweise keine Benutzerverwaltung mit einem Rollenkonzept implementiert. Bei einem vermarktbaren Produkt wären solche Funktionalitäten zur Verfügung zu stellen. Der wissenschaftliche Anspruch sowie der Erkenntnisgewinn wäre bei der Integration solcher Komponenten jedoch vergleichbar gering bis nicht vorhanden. Hierfür kursieren unzählige Beispiele im Internet. Wie der Titel der Arbeit verlauten lässt, soll das System einen als gut zu bewertenden Prototyp darstellen. Sie erhebt nicht den Anspruch einer vollumfassenden Softwarelösung, die für den Verkauf geeignet ist. Zukünftige Weiterentwicklungen könnten diese Aspekte berücksichtigen.

Die Generierung von textbasierten Übungsaufgaben könnte durch eine Erweiterung des Trainingsmaterials in neuartige Modelle wie beispielsweise ChatGPT integriert werden. Ein großer Vorteil hierbei wäre, dass das Modell auch gleichzeitig auf die Beantwortung von Fragen der Studierenden

trainierbar wäre. Ob simultan auch die Erstellung einer Musterlösung durch dieses sinnvoll ist, gilt es zu evaluieren. Kontextabhängig sind beschriebene Diagrammkomponenten unterschiedlich zu modellieren.

Auch eine Untersuchung der Effektivität von Echtzeit-Feedback während des Modellierungsvorgangs ist von Interesse. Bisher ist unklar, ob dieses für Lernende nutzbringend oder störend ist. Das präsentierte System sieht eine Weiterentwicklung diesbezüglich vor und ist dahingehend mit minimalem Aufwand erweiterbar.

In der Evaluation nannten die Studierenden unterschiedliche Ideen, die ihnen für einen Ausbau des Funktionsumfangs einfielen. Dazu zählen (durch den Autor aufgrund von Wiederholungen verkürzt dargestellt):

- Mehr textbasierte Beschreibungen respektive Informationen zu dargestellten Komponenten.
- Erweiterung des Chats.
- Mehr Lehrvideos.
- Speicherung von Übungsaufgaben, die in der Rubrik für die Kollaboration eingetragen wurden.
- Verknüpfung des Feedbacks mit einem Skript (dieser Zusammenhang wäre jedoch bei einer Änderung des Skripts immer wieder neu herzustellen).
- Rubrik, in der Studierende ihren Lernfortschritt verfolgen können.
- Auswertungen für Studierende, die beispielsweise anhand ihres Fehlerprofils über die Zeit darstellen, wo sie die meisten Fehler machen.
- Lerntipps, die anhand des Fehlerprofils von einem Machine- oder Deep-Learning-Modell erzeugt werden.

Zusätzlich ist eine Erweiterung der Aufgabentypen möglich. Aktuell fokussiert sich das System auf die Erstellung textueller Übungsaufgaben mit Beschreibungen für UML-Klassendiagramme. So wäre es beispielsweise möglich, es um Schritte der Continuous Practices (vgl. Huber und Hagel, 2021) zu erweitern und so einen größeren Teil des gesamten Softwareentwicklungszyklus abzudecken sowie dadurch weitere Inhalte des Fachbereichs Software

Engineering zu integrieren. Die Einbindung neuer Übungsaufgaben für Entwurfsmuster ist außerdem denkbar. Auch die Rubrik Tutorial könnte diesbezüglich um Lehrmaterial und -videos erweitert werden. Außerdem wären Aufgaben zu allen weiteren UML-Diagrammtypen vorstellbar. Das System ist aktuell nur auf Klassendiagramme ausgelegt. Eine solche Erweiterung wäre jedoch mit vergleichbarem Vorgehen abbildbar.

Eine Anwendung des Systems in agilen Lehr- und Lernarrangements, wie zum Beispiel von Müller-Amthor u. a. (2020) vorgestellt, ist ebenfalls denkbar. Wie mögliche Weiterentwicklungen aussehen müssen, ist jedoch vorab zu evaluieren.

Auch eine Symbiose der computergestützten Hilfestellung mit einer spielbasierten Lernumgebung, wie beispielsweise von Gensheimer u. a. (2020) präsentiert, ist denkbar. Die Autoren versuchen unter Verwendung sogenannter Visual Novels spielend Lehrinhalte zu vermitteln (vgl. Gensheimer u. a., 2020).

Letztlich könnte noch ein Lehrvideo für den Einsatz des Systems zur Verfügung gestellt werden. So ist sicherzustellen, dass die Studierende alle Funktionalitäten nachvollziehen und optimal für sich einsetzen können, um ihren Lernerfolg zu verbessern.

ANHANG A

Anhang

A.1 Mögliche Aktivitäten von Akteuren in einem Lehrkontext

Die für dieses Kapitel zugehörige Beschreibung findet sich in Abschnitt 3.2.

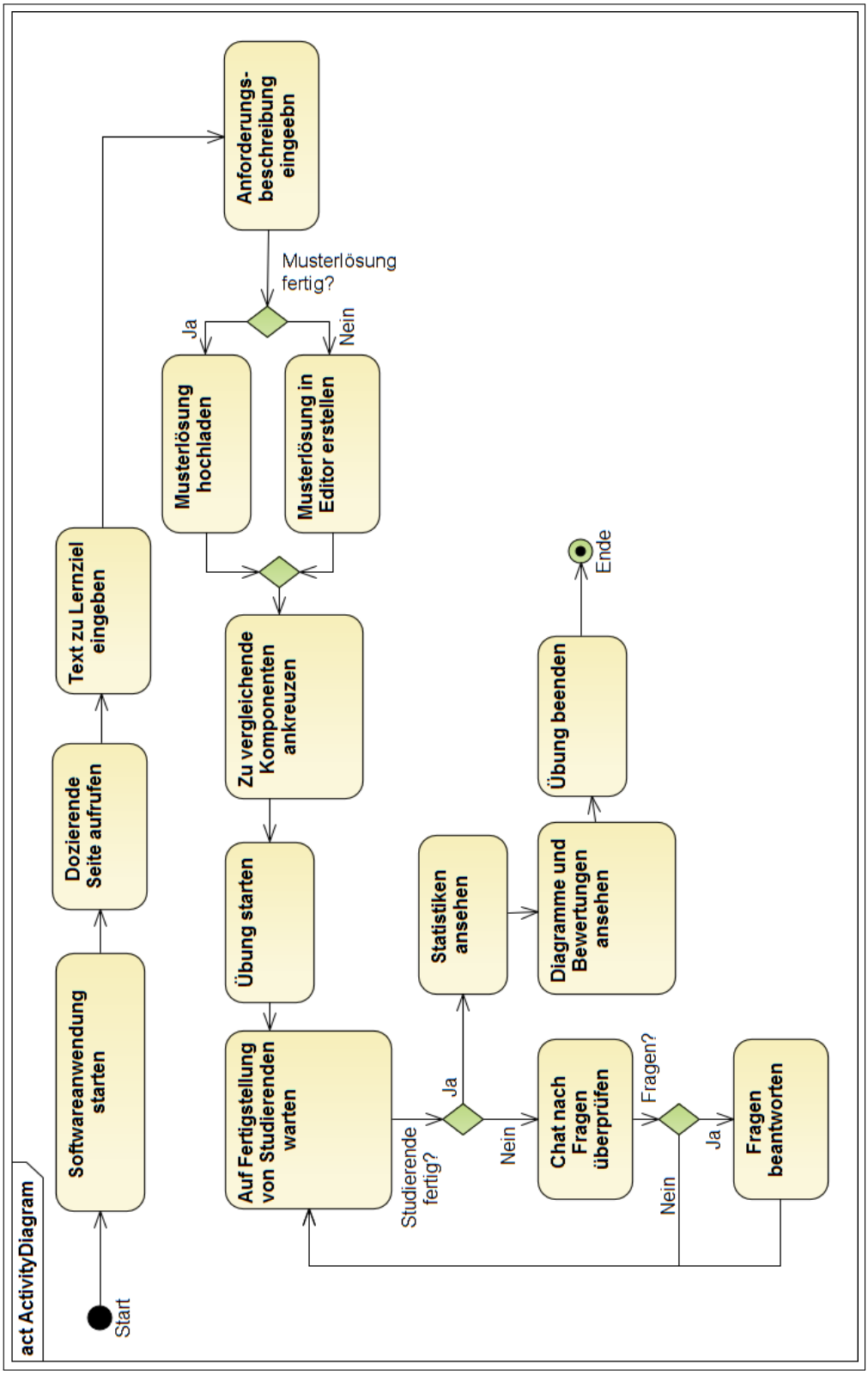


Abbildung A.1: Von Anforderungen abgeleitete Aktivitäten von Dozierenden in einer zu entwickelnden Softwareanwendung. Eigene Darstellung in Anlehnung an (Huber u. a., 2023a, S. 187).

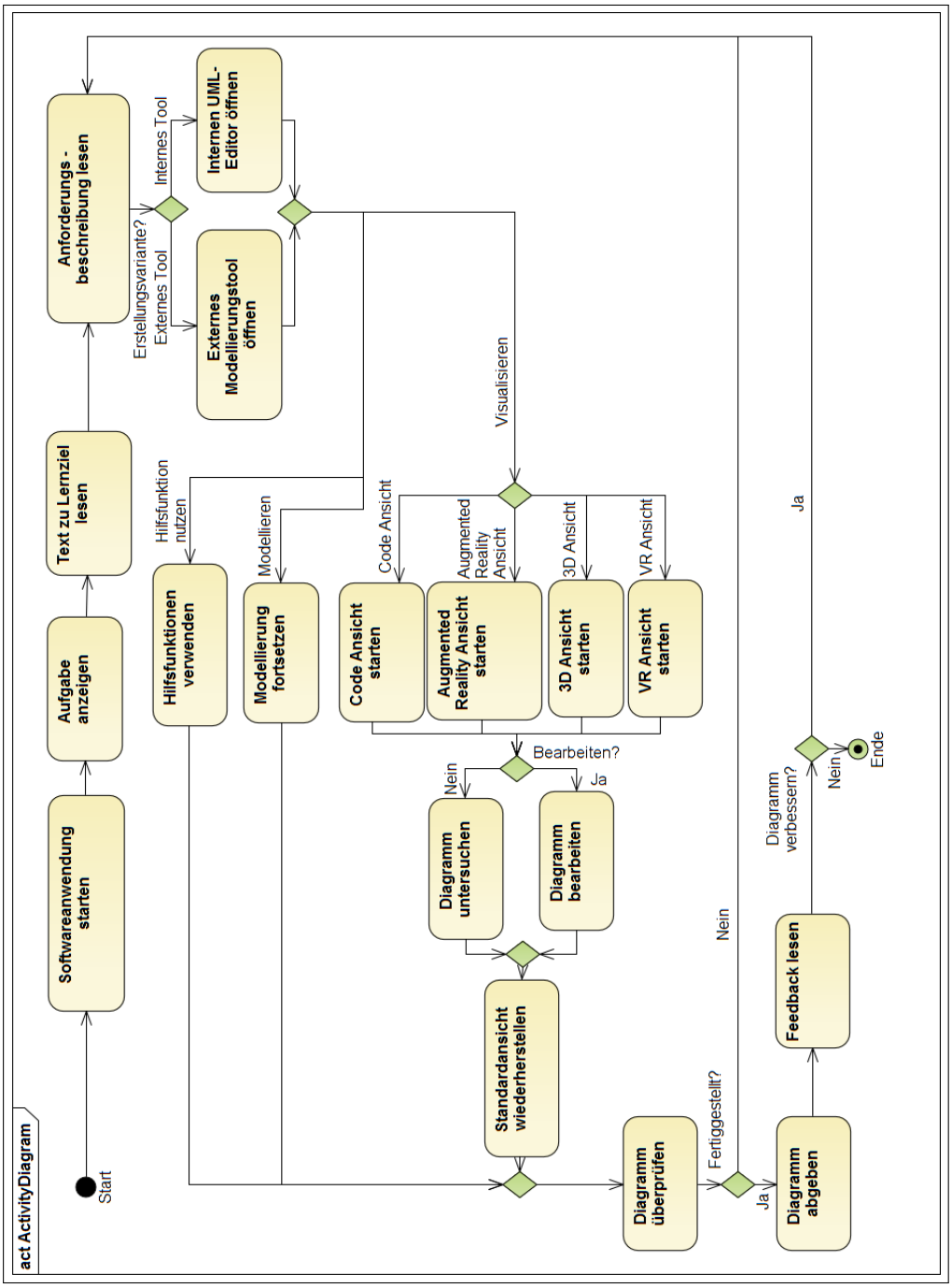


Abbildung A.2: Von Anforderungen abgeleitete Aktivitäten von Studierenden in einer zu entwickelnden Softwareanwendung. Eigene Darstellung in Anlehnung an (Huber u. a., 2023a, S. 187).

A.2 Leitfaden für die qualitative Anforderungserhebung von Dozierenden an eine softwaregestützte Hilfestellung für die Lehre von UML-Diagrammen in einem hochschulischen Kontext

Die folgenden Seiten beinhalten den Leitfaden, der für die qualitative Anforderungserhebung von Dozierenden an eine softwaregestützte Hilfestellung für die Lehre von UML-Diagrammen in einem hochschulischen Kontext verwendet wurde.

Leitfaden für ein qualitatives Experteninterview mit Lehrenden zu ihren Anforderungen für eine Tool-gestützte Lehre von UML-Diagrammen

1 Einleitungs- bzw. Vorstellungsphase

1.1 Dank für Gesprächsbereitschaft

Vielen Dank, dass Sie sich für die Teilnahme an diesem Experteninterview entschieden haben.

1.2 Vorstellung des Interviewers und institutioneller Kontext

Kurze Vorstellung des Interviewers.

1.3 Erläuterung des Themas der eigenen Untersuchung

Softwaresysteme und Algorithmen finden zunehmen Einzug und Anwendung in der hochschulischen Lehre. So auch im Software Engineering. Bei dieser Untersuchung handelt es sich um ein leitfadengestütztes Experteninterview. Sie werden in diesem Kontext als Experte zu Ihrer Meinung und Ihren Erfahrungen befragt. Inhaltlich soll untersucht werden, welche Anforderungen Sie als Expert*in an eine softwaregestützte Hilfestellung zur Lehre von UML-Diagrammen (im Kontext der hochschulischen Lehre von Software Engineering) erheben.

1.4 Erläuterung des Interviewablaufs

Das Interview beinhaltet unterschiedliche Themenschwerpunkte, die sich an den zu beantwortenden Forschungsfragen orientieren. Nach einer kurzen Einführungsfrage können Sie sich zu drei unterschiedlichen Themenschwerpunkten äußern. Diese sind in drei Hauptfragen mit jeweiligen Unterfragen gegliedert. Das Experteninterview ist auf ca. 30 Minuten ausgelegt. Die Ergebnisse aller Interviews können publiziert werden. Ihre Angaben werden anonymisiert und konsolidiert. Diese sind nicht auf Sie zurückzuführen.

1.5 Tonaufzeichnung

□ Anonymitätszusicherung gegeben

□ Erlaubnis zur Tonaufzeichnung erhalten

2 Befragungsphase

2.1 Persönliche Angaben

1. **Wie viele Semester lehren Sie bereits Software Engineering?** Seit _____ Semestern.

2.2 Themenschwerpunkt I: Mit welchen Herausforderungen sehen sich die Studierenden eines Software Engieering Kurses Ihrer Meinung nach konfrontiert, wenn es um das Erlernen der UML und die Erstellung zugehöriger Diagramme geht?

2.2.1 Vor welche Herausforderungen stellt Studierende das Erlernen Objekt-Orientierter Konzepte?

2.2.2 Vor welche Herausforderungen stellt Studierende das Anwenden Objekt-Orientierter Konzepte in Ihren Modellierungen von UML-Diagrammen?

2.2.3 Vor welche Herausforderungen stellt es Studierende, sich an die erlernten Diagrammkomponenten (z.B. Klassen, Interfaces) zu erinnern?

2.2.4 Vor welche Herausforderungen stellt es Studierende, alle in einer textuellen Aufgabe enthaltenen Zusammenhänge zu identifizieren und alle nötigen Diagrammkomponenten zu modellieren?

2.2.5 Welches sind Ihrer Meinung nach die gravierendsten Herausforderungen für Studierende bei der Erstellung von UML-Diagrammen anhand textueller Aufgaben?

2.2.6 Haben Studierende im Laufe des von Ihnen angebotenen Software Engineering Kurses Zugriff auf ein Tool, dass sie bei der Modellierung von UML-Diagrammen anhand textueller Aufgaben unterstützt?

2.2.7 Geben Sie all Ihren Studierenden regelmäßiges und individuelles Feedback?

2.2.8 Würde sich individuelles und direktes Feedback Ihrer Meinung nach positiv auf den Lernerfolg Ihrer Studierenden auswirken?

2.2.9 Sind Sie der Meinung, dass zusätzliche Übungsangebote in Form textueller Aufgaben für den Lernerfolg der Studierenden hilfreich wären?

2.2.10 Sind Sie der Meinung, dass die Verfügbarkeit einer Musterlösung zu einer gestellten textuellen Übungsaufgabe für den Lernerfolg der Studierenden wichtig ist?

2.3 Themenschwerpunkt II: Welche softwaregestützten Hilfestellungen könnten Studierende von der Bereitstellung einer Aufgabe bishin zu individuellem Feedback unterstützen?

2.3.1 Wie könnte Studierende das automatische Erstellen textueller Übungsaufgaben für UML-Diagramme Ihrer Meinung nach in ihrem Lernprozess unterstützen?

2.3.2 Wie könnte Studierende das automatische Generieren von Musterlösungen zu textuellen Aufgaben Ihrer Meinung nach in ihrem Lernprozess unterstützen?

2.3.3 Wie könnte Ihrer Meinung nach individuelles und direktes Feedback zu den von Studierenden erstellten UML-Diagrammen durch eine Software den Lernerfolg verbessern?

2.3.4 Wie sollte ein solches Feedback Ihrer Meinung nach gestaltet sein?

2.4 Themenschwerpunkt III: Welche konkreten Funktionalitäten müssen für Sie als Lehrperson bereitgestellt werden, um eine Übungsstunde in diesem Kontext digital abzubilden?

2.4.1 Welche Anforderungen müssen Ihrer Meinung nach für Lehrpersonen erfüllt sein, damit Sie diese Anwendung einsetzen würden?

3 Abschlussphase

3.1 Danksagung

Vielen Dank, dass Sie sich für die Teilnahme an diesem Experteninterview Zeit genommen haben!

3.2 Feedback

Folgende Punkte würden uns im Nachgang zu den gestellten Fragen interessieren:

- Wurden Ihrer Meinung nach Punkte vergessen, die in diesem Kontext relevant sind?
- Hätten Sie noch etwas ergänzt?

3.3 Schneeballprinzip

Können Sie einen Ihrer Kolleg:innen von anderen Hochschulen oder Universitäten benennen, die ebenfalls hilfreiche Inhalte zu den obigen Fragen beitragen könnten?

A.3 Fragebogen für die Evaluation des konzipierten und prototypisch implementierten Systems

Die folgenden Seiten beinhalten den Fragebogen, der für die Evaluation des konzipierten und prototypisch implementierten Systems zum Einsatz kam. Die zugrundeliegende Literatur, anhand derer er entwickelt wurde, ist innerhalb des Fließtextes der Arbeit beschrieben.

Fragebogen zur vorgestellten, prototypischen Software für die Unterstützung der Lehre von UML-Klassendiagrammen in der hochschulischen Lehre von Software Engineering

Liebe Studierende, vielen Dank, dass Sie sich die Zeit für diesen Fragebogen nehmen. Im Folgenden möchte ich Sie zu der Ihnen vorgestellten prototypischen Software befragen.

Bitte lesen Sie alle Fragen gründlich und beantworten Sie diese im Anschluss. Sofern es bei einer Frage mehrere Antwortmöglichkeiten gibt, wählen Sie bitte eine passende aus. Es gibt bei diesen geschlossenen Fragen keine Mehrfachantworten.

Hinweis: Im Folgenden ist wiederholt von dem Begriff "Benutzeroberfläche"die Rede. Im Kontext des Fragebogens ist dieser als sichtbare Visualisierung bereitgestellter Inhalte des softwaregestützten Prototyps definiert und stellt somit die Schnittstelle zwischen Benutzer*in und der zugrundeliegenden Software dar. Der Begriff verweist hier weiterhin auf alle aufrufbaren Seiten.

Persönliche Angaben

Wie alt sind Sie? Ich bin _____ Jahre alt.

Bitte geben Sie ihr Geschlecht an.
- ☐ Männlich
- ☐ Weiblich
- ☐ Divers
- ☐ Keine Angabe

Bitte geben Sie Ihren Studiengang an.
- ☐ Informatik
- ☐ Informatik - Game Engineering
- ☐ Anderer Studiengang: _____

Allgemeine Fragen zu der Benutzeroberfläche

Wie beurteilen Sie die folgenden Aussagen?	**Stimme nicht zu**	**Stimme eher nicht zu**	**Stimme eher zu**	**Stimme voll zu**	**Kann ich nicht beurteilen**
Die grafische Gestaltung der Benutzeroberfläche empfinde ich als ansprechend.	☐	☐	☐	☐	☐
Die Benutzeroberfläche finde ich übersichtlich.	☐	☐	☐	☐	☐
Innerhalb der Benutzeroberfläche finde ich mich intuitiv zurecht.	☐	☐	☐	☐	☐

Die Benutzeroberfläche nehme ich als selbsterklärend wahr.	☐	☐	☐	☐	☐
Die Benutzeroberfläche bewerte ich für den Einsatz in der hochschulischen Lehre als angemessen.	☐	☐	☐	☐	☐
Den strukturellen Aufbau der Benutzeroberfläche bewerte ich als gut.	☐	☐	☐	☐	☐

Wie würden Sie die Benutzeroberfläche insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

0	1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Bezogen auf die Benutzeroberfläche hätte ich mir diese Funktionalitäten oder Darstellungen zusätzlich gewünscht:

Weiteres Feedback zu der Benutzeroberfläche:

Fragen zu dem Programmbereich „Tutorial”

Wie beurteilen Sie die folgenden Aussagen?	**Stimme nicht zu**	**Stimme eher nicht zu**	**Stimme eher zu**	**Stimme voll zu**	**Kann ich nicht beurteilen**
Die Inhalte des Programmbereichs Tutorial bewerte ich als lehrreich.	☐	☐	☐	☐	☐
Den strukturellen Aufbau des Programmbereichs Tutorial bewerte ich als sinnvoll.	☐	☐	☐	☐	☐

Ich kann mir vorstellen, den Programmbereich Tutorial für den Aufbau bzw. die Erweiterung meines Wissens zu verwenden.	☐	☐	☐	☐	☐
Die beispielhafte Übungsaufgabe empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die beispielhafte Lösung für die bereitgestellte Übungsaufgabe empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Den Schwierigkeitsgrad der dargestellten Übungsaufgabe bewerte ich als angemessen.	☐	☐	☐	☐	☐
Die dargestellte Übungsaufgabe empfinde ich als hilfreich.	☐	☐	☐	☐	☐

Wie würden Sie den Programmbereich Tutorial insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

0	1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Bezogen auf das Tutorial hätte ich mir diese Funktionalitäten oder Darstellungen zusätzlich gewünscht:

Weiteres Feedback zum Tutorial:

Fragen zu dem Programmbereich „Studierende"

Wie beurteilen Sie die folgenden Aussagen?	**Stimme nicht zu**	**Stimme eher nicht zu**	**Stimme eher zu**	**Stimme voll zu**	**Kann ich nicht beurteilen**
Den strukturellen Aufbau des Programmbereichs Studierende bewerte ich als sinnvoll.	☐	☐	☐	☐	☐

Die Beschreibung des Lernziels enthält meiner Meinung nach alle wichtigen Aspekte.	☐	☐	☐	☐	☐
Die verfügbaren Themengebiete für die automatisierte Erstellung einer Übungsaufgabe bewerte ich als ausreichend.	☐	☐	☐	☐	☐
Die verfügbaren Schwierigkeitsgrade für die automatisierte Erstellung einer Übungsaufgabe bewerte ich als ausreichend.	☐	☐	☐	☐	☐
Die Hilfestellung, welche die Diagrammbestandteile einer textuellen Anforderungsbeschreibung anzeigt, bewerte ich als gut verständlich.	☐	☐	☐	☐	☐
Die Inhalte der angezeigten Strategiehilfe empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Anzeige des Vorlesungsskripts empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Das Hochladen eines UML-Klassendiagramms hat gut funktioniert.	☐	☐	☐	☐	☐
Die Chatfunktion empfinde ich als hilfreich.	☐	☐	☐	☐	☐

Wie würden Sie den Programmbereich Studierende insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

0	1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Bezogen auf den Programmbereich Studierende hätte ich mir diese Funktionalitäten oder Darstellungen zusätzlich gewünscht:

Weiteres Feedback zu dem Programmbereich Studierende:

Fragen zu dem Programmbereich „Kollaboration”

Wie beurteilen Sie die folgenden Aussagen?	Stimme nicht zu	Stimme eher nicht zu	Stimme eher zu	Stimme voll zu	Kann ich nicht beurteilen
Den strukturellen Aufbau des Programmbereichs Kollaboration bewerte ich als sinnvoll.	☐	☐	☐	☐	☐
Die Erstellung einer eigenen Übungsaufgabe funktioniert problemlos.	☐	☐	☐	☐	☐
Die automatisierte Erstellung einer Übungsaufgabe funktioniert problemlos.	☐	☐	☐	☐	☐
Die Auswahlmöglichkeiten bezogen auf das Themengebiet bei der automatisierten Erstellung einer Übungsaufgabe empfinde ich als ausreichend.	☐	☐	☐	☐	☐
Die Auswahlmöglichkeiten bezogen auf den Schwierigkeitsgrad bei der automatisierten Erstellung einer Übungsaufgabe empfinde ich als ausreichend.	☐	☐	☐	☐	☐
Das Hochladen einer Musterlösung hat problemlos funktioniert.	☐	☐	☐	☐	☐
Das automatisierte Erstellen einer Musterlösung hat problemlos funktioniert.	☐	☐	☐	☐	☐

Wie würden Sie den Programmbereich Kollaboration insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

0	1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Bezogen auf den Programmbereich Kollaboration hätte ich mir diese Funktionalitäten oder Darstellungen zusätzlich gewünscht:

Weiteres Feedback zu dem Programmbereich Kollaboration:

Fragen zu dem erhaltenen Feedback

Wie beurteilen Sie die folgenden Aussagen?	Stimme nicht zu	Stimme eher nicht zu	Stimme eher zu	Stimme voll zu	Kann ich nicht beurteilen
Die Rückmeldung zu fehlenden Diagrammkomponenten habe ich problemlos verstanden.	☐	☐	☐	☐	☐
Anhand der Rückmeldung zu fehlenden Diagrammkomponenten kann ich mir erklären, wo mir bei der Modellierung ein Fehler unterlaufen ist.	☐	☐	☐	☐	☐
Die Rückmeldung zu überflüssigen Diagrammkomponenten habe ich problemlos verstanden.	☐	☐	☐	☐	☐
Anhand der Rückmeldung zu überflüssigen Diagrammkomponenten kann ich mir erklären, wo mir bei der Modellierung ein Fehler unterlaufen ist.	☐	☐	☐	☐	☐
Die Rückmeldung zu gleichnamigen Diagrammkomponenten mit unterschiedlichen Attributen, Methoden oder Relationen habe ich problemlos verstanden.	☐	☐	☐	☐	☐
Anhand der Rückmeldung zu gleichnamigen Diagrammkomponenten mit unterschiedlichen Attributen, Methoden oder Relationen kann ich mir erklären, wo mir bei der Modellierung ein Fehler unterlaufen ist.	☐	☐	☐	☐	☐

Die Rückmeldung der zu erkennenden Elemente in der textuellen Anforderungsbeschreibung empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Tipps zu den fehlenden Klassen innerhalb des studentischen UML-Klassendiagramms empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Tipps zu den überflüssigen Klassen innerhalb des studentischen UML-Klassendiagramms empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Tipps zu den Attributen empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Tipps zu den Methoden empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Tipps zu den Beziehungen der Diagrammkomponenten untereinander empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Tipps zu den zu wiederholenden Kursinhalten empfinde ich als hilfreich.	☐	☐	☐	☐	☐
Die Empfehlung zum weiteren Vorgehen empfinde ich als hilfreich.	☐	☐	☐	☐	☐

Wie würden Sie das erhaltene Feedback insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

0	1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Bezogen auf das Feedback hätte ich mir diese Funktionalitäten oder Darstellungen zusätzlich gewünscht:

Weiteres Feedback zum dem erhaltenen Feedback:

Allgemeine Fragen zu dem softwaregestützten Prototyp

Wie beurteilen Sie die folgenden Aussagen?	**Stimme nicht zu**	**Stimme eher nicht zu**	**Stimme eher zu**	**Stimme voll zu**	**Kann ich nicht beurteilen**
Ich würde den Prototypen verwenden, um mein Wissen zu UML-Klassendiagrammen auf- oder auszubauen.	☐	☐	☐	☐	☐
Ich würde den Prototypen verwenden, um mich auf Prüfungsaufgaben zu UML-Klassendiagrammen vorzubereiten.	☐	☐	☐	☐	☐
Meinen Kommiliton*innen würde ich die Nutzung des Prototyps empfehlen.	☐	☐	☐	☐	☐

Wie würden Sie den Funktionsumfang des Prototyps insgesamt auf einer Skala von 0 bis 10, wobei 0 sehr schlecht und 10 sehr gut bedeutet, bewerten?

0	1	2	3	4	5	6	7	8	9	10
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Diese Funktionalitäten oder Darstellungen hätte ich mir zusätzlich gewünscht:

Weiteres Feedback

Weiteres Feedback zu dem vorgestellten Softwareprototypen:

Vielen Dank für Ihr Engagement und Ihre Teilnahme!

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CNN Convolutional Neural Network

CSPNet Cross Stage Partial Network

DRY Don't Repeat Yourself

DSR Design Science Research

EAST Efficient and Accuracy Scene Text

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

ISR Information Systems Research

JSON JavaScript Object Notation

KNN Künstliches Neuronales Netz

LSTM Long Short-Term Memory

MLP Multi Layer Perzeptron

NLP Natural Language Processing

OMG Object Management Group

OMT Object Modeling Technique

OOSE Object-Oriented Software Engineering

PANet Path Aggregation Network

REST Representational State Transfer

RNN Rekurrentes Neuronales Netz

SLR Systematische Literaturübersicht

UML Unified Modeling Language

XMI XML Metadata Interchange

XML Extensible Markup Language

YOLO You Only Look Once

Codebeispielverzeichnis

Literaturverzeichnis

Abdelnabi, Esra A., Abdelsalam M. Maatuk und Mohammed Hagal (2021). „Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques“. In: *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*, S. 288–293. DOI: 10.1109/MI-STA52233.2021.9464433.

Aggarwal, Charu C. (2018). „Neural Networks and Deep Learning: A Textbook“. In: Cham: Springer International Publishing. ISBN: 978-3-319-94463-0. DOI: 10.1007/978-3-319-94463-0. URL: https://doi.org/10.1007/978-3-319-94463-0.

Ahmed, Sharif, Arif Ahmed und Nasir U. Eisty (2022). „Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review“. In: *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*, S. 112–119. DOI: 10.1109/SERA54885.2022.9806783.

Alexander, Christopher u. a. (1977). *A pattern language: towns, buildings, construction*. Oxford university press.

Alhazmi, Sohail, Charles Thevathayan und Margaret Hamilton (2021). „Learning UML Sequence Diagrams with a New Constructivist Pedagogical Tool: SD4ED“. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE '21. Virtual Event, USA: Association for Computing Machinery, S. 893–899. ISBN: 9781450380621.

DOI: 10.1145/3408877.3432521. URL: https://doi.org/10.1145/3408877.3432521.

Ali, Norhayati Mohd u. a. (2018). „UML Diagram Learning Tool“. In: *University Carnival on e-learning (IUCEL) 2018*, S. 408.

Azevedo, Frederico A.C. u. a. (2009). „Equal numbers of neuronal and non-neuronal cells make the human brain an isometrically scaled-up primate brain“. In: *Journal of Comparative Neurology* 513.5, S. 532–541. DOI: https://doi.org/10.1002/cne.21974. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cne.21974. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cne.21974.

Or-Bach, Rachel und Ilana Lavy (Juni 2004). „Cognitive Activities of Abstraction in Object Orientation: An Empirical Study“. In: *SIGCSE Bull.* 36.2, S. 82–86. ISSN: 0097-8418. DOI: 10.1145/1024338.1024378. URL: https://doi.org/10.1145/1024338.1024378.

Bank, Dor, Noam Koenigstein und Raja Giryes (2020). *Autoencoders*. DOI: 10.48550/ARXIV.2003.05991. URL: https://arxiv.org/abs/2003.05991.

Bargh, John A. u. a. (2001). „The automated will: Nonconscious activation and pursuit of behavioral goals.“ en. In: *Journal of Personality and Social Psychology* 81.6, S. 1014–1027. DOI: 10.1037/0022-3514.81.6.1014. URL: http://dx.doi.org/10.1037/0022-3514.81.6.1014.

Baur, Nina und Jörg Blasius (2019). *Handbuch Methoden der empirischen Sozialforschung*. Wiesbaden: Springer Fachmedien Wiesbaden. ISBN: 978-3-658-21308-4. DOI: 10.1007/978-3-658-21308-4_1. URL: https://doi.org/10.1007/978-3-658-21308-4_1.

Beck, Henning, Sofia Anastasiadou und Christopher Meyer zu Reckendorf (2018). „Neurone in Aktion“. In: *Faszinierendes Gehirn: Eine bebilderte Reise in die Welt der Nervenzellen*. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 140–221. ISBN: 978-3-662-54756-4. DOI: 10.1007/978-3-662-54756-4_4. URL: https://doi.org/10.1007/978-3-662-54756-4_4.

Bézivin, Jean und Pierre-Alain Muller (1999). „UML: The Birth and Rise of a Standard Modeling Notation“. In: *The Unified Modeling Language. «UML»'98: Beyond the Notation*. Hrsg. von Jean Bézivin und Pierre-

Alain Muller. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 1–8. ISBN: 978-3-540-48480-6.

Bian, Weiyi, Omar Alam und Jörg Kienzle (2019). „Automated Grading of Class Diagrams“. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, S. 700–709. DOI: 10.1109/MODELS-C.2019.00106.

— (2021). „Automated Grading of Class Diagrams“. In: *Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems*. MODELS '19. Munich, Germany: IEEE Press, S. 700–709. ISBN: 9781728151250. DOI: 10.1109/MODELS-C.2019.00106. URL: https://doi.org/10.1109/MODELS-C.2019.00106.

Bochkovskiy, Alexey, Chien-Yao Wang und Hong-Yuan Mark Liao (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv: 2004.10934 [cs.CV].

Bogdanova, Daria (2019). „Towards Personalized Feedback in a Smart Learning Environment For Teaching Conceptual Modelling“. In: *2019 13th International Conference on Research Challenges in Information Science (RCIS)*, S. 1–5. DOI: 10.1109/RCIS.2019.8876983.

Bogner, Alexander, Beate Littig und Wolfgang Menz (2014). „Interviews mit Experten: Eine praxisorientierte Einführung“. In: Wiesbaden: Springer Fachmedien Wiesbaden. ISBN: 978-3-531-19416-5. DOI: 10.1007/978-3-531-19416-5. URL: https://doi.org/10.1007/978-3-531-19416-5.

Bolloju, Narasimha und Felix S.K. Leung (Juli 2006). „Assisting Novice Analysts in Developing Quality Conceptual Models with UML“. In: *Commun. ACM* 49.7, S. 108–112. ISSN: 0001-0782. DOI: 10.1145/1139922.1139926. URL: https://doi.org/10.1145/1139922.1139926.

Booch, Grady, James Rumbaugh und Ivar Jacobson (1998). „The unified modeling language user guide“. In: *Reading: Addison-Wesley*.

Boubekeur, Younes, Gunter Mussbacher und Shane McIntosh (2020). „Automatic Assessment of Students' Software Models Using a Simple Heuristic and Machine Learning“. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. New York, NY, USA: Association for

Computing Machinery. ISBN: 9781450381352. URL: https://doi.org/10.1145/3417990.3418741.

Bourque, Pierre und Richard E. Fairley, Hrsg. (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0. Los Alamitos, CA: IEEE Computer Society. ISBN: 978-0-7695-5166-1. URL: http://www.swebok.org/.

Bracken, Barbara (Dez. 2003). „Progressing from Student to Professional: The Importance and Challenges of Teaching Software Engineering“. In: *J. Comput. Sci. Coll.* 19.2, S. 358–368. ISSN: 1937-4771.

Brereton, Pearl u. a. (2007). „Lessons from applying the systematic literature review process within the software engineering domain“. In: *Journal of Systems and Software* 80.4. Software Performance, S. 571–583. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2006.07.009. URL: http://www.sciencedirect.com/science/article/pii/S016412120600197X.

Brown, Tom B. u. a. (2020). „Language Models Are Few-Shot Learners“. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc. ISBN: 9781713829546.

Budgen, David u. a. (Jan. 2008). „Using Mapping Studies in Software Engineering“. In: *Proceedings of PPIG 2008* 2.

Cai, Yuanfang, Daniel Iannuzzi und Sunny Wong (2011). „Leveraging design structure matrices in software design education“. In: *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, S. 179–188. DOI: 10.1109/CSEET.2011.5876085.

Cai, Yuanfang u. a. (2013). „Introducing tool-supported architecture review into software design education“. In: *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*, S. 70–79. DOI: 10.1109/CSEET.2013.6595238.

Cavalcanti, Anderson Pinheiro u. a. (2021). „Automatic feedback in online learning environments: A systematic literature review“. In: *Computers and Education: Artificial Intelligence* 2, S. 100027. ISSN: 2666-920X. DOI: https://doi.org/10.1016/j.caeai.2021.100027. URL: https://www.sciencedirect.com/science/article/pii/S2666920X21000217.

Chen, Fangwei u. a. (2022). „Automatically recognizing the semantic elements from UML class diagram images“. In: *Journal of Systems and Software* 193, S. 111431. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111431. URL: https://www.sciencedirect.com/science/article/pii/S0164121222001340.

Chen, Lianipng, Muhammad Ali Babar und He Zhang (2010). „Towards an Evidence-Based Understanding of Electronic Data Sources“. In: *14th International Conference on Evaluation and Assessment in Software Engineering (EASE) (EASE)*. URL: https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/EASE2010.17.

Chin, John P., Virginia A. Diehl und Kent L. Norman (1988). „Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface“. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '88. Washington, D.C., USA: Association for Computing Machinery, S. 213–218. ISBN: 0201142376. DOI: 10.1145/57167.57203. URL: https://doi.org/10.1145/57167.57203.

Chren, Stanislav u. a. (2019). „Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course“. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, S. 100–109. DOI: 10.1109/ICSE-SEET.2019.00019.

Coburn, Joseph (2020). „Getting Started with Flask“. In: *Build Your Own Car Dashboard with a Raspberry Pi: Practical Projects to Build Your Own Smart Car*. Berkeley, CA: Apress, S. 143–170. ISBN: 978-1-4842-6080-7. DOI: 10.1007/978-1-4842-6080-7_6. URL: https://doi.org/10.1007/978-1-4842-6080-7_6.

Cornelsen Verlag GmbH (2023a). *ähnlich — duden.de*. https://www.duden.de/rechtschreibung/aehnlich. [Accessed 06-10-2023].

— (2023b). *identisch — duden.de*. https://www.duden.de/rechtschreibung/identisch. [Accessed 06-10-2023].

Denning, Peter J. (Feb. 1997). „A New Social Contract for Research“. In: *Commun. ACM* 40.2, S. 132–134. ISSN: 0001-0782. DOI: 10.1145/253671.253755. URL: https://doi.org/10.1145/253671.253755.

Didier, Jean-Pierre und Emmanuel Bigand (2011). *Rethinking physical and rehabilitation medicine: New technologies induce new learning strategies*.

Paris: Springer Paris. ISBN: 978-2-8178-0034-9. DOI: 10.1007/978-2-8178-0034-9. URL: https://doi.org/10.1007/978-2-8178-0034-9.

Dingsøyr, Torgeir (2021). „Agile Iteration Reviews in a Project Course: A key to Improved Feedback and Assessment Practice“. In: *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*, S. 21–25. DOI: 10.1109/SEENG53126.2021.00011.

Döring, Nicola und Jürgen Bortz (2016). *Forschungsmethoden und Evaluation in den Sozial- und Humanwissenschaften*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-41089-5. DOI: 10.1007/978-3-642-41089-5. URL: https://doi.org/10.1007/978-3-642-41089-5.

Dörn, Sebastian (2018). „Neuronale Netze“. In: *Programmieren für Ingenieure und Naturwissenschaftler: Intelligente Algorithmen und digitale Technologien*. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 89–148. ISBN: 978-3-662-54304-7. DOI: 10.1007/978-3-662-54304-7_3. URL: https://doi.org/10.1007/978-3-662-54304-7_3.

Du, Ke-Lin und M. N. S. Swamy (2014). „Introduction“. In: *Neural Networks and Statistical Learning*. London: Springer London, S. 1–14. ISBN: 978-1-4471-5571-3. DOI: 10.1007/978-1-4471-5571-3_1. URL: https://doi.org/10.1007/978-1-4471-5571-3_1.

Duschl, Kerstin, Martin Obermeier und Birgit Vogel-Heuser (2014). „An experimental study on UML Modeling errors and their causes in the education of model driven PLC programming“. In: *2014 IEEE Global Engineering Education Conference (EDUCON)*, S. 119–128. DOI: 10.1109/EDUCON.2014.6826078.

Eigler, Tobias, Florian Huber und Georg Hagel (2023). „Tool-Based Software Engineering Education for Software Design Patterns and Software Architecture Patterns - a Systematic Literature Review“. In: *Proceedings of the 5th European Conference on Software Engineering Education*. ECSEE '23. Seeon/Bavaria, Germany: Association for Computing Machinery, S. 153–161. ISBN: 9781450399562. DOI: 10.1145/3593663.3593670. URL: https://doi.org/10.1145/3593663.3593670.

Eilebrecht, Karl und Gernot Starke (2019). *Patterns kompakt: Entwurfsmuster für effektive Softwareentwicklung*. Berlin, Heidelberg: Springer

Berlin Heidelberg. ISBN: 978-3-662-57937-4. DOI: 10.1007/978-3-662-57937-4. URL: https://doi.org/10.1007/978-3-662-57937-4.

Estes, Cheryl A. (2004). „Promoting Student-Centered Learning in Experiential Education“. In: *Journal of Experiential Education* 27.2, S. 141–160. DOI: 10.1177/105382590402700203. eprint: https://doi.org/10.1177/105382590402700203. URL: https://doi.org/10.1177/105382590402700203.

Evans, Carol (2013). „Making sense of assessment feedback in higher education“. In: *Review of educational research* 83.1, S. 70–120.

Farah, Juan Carlos u. a. (2022). „Impersonating Chatbots in a Code Review Exercise to Teach Software Engineering Best Practices“. In: *2022 IEEE Global Engineering Education Conference (EDUCON)*, S. 1634–1642. DOI: 10.1109/EDUCON52537.2022.9766793.

Farias, Kleinner und Bruno C. da Silva (2020). „What's the Grade of Your Diagram? Towards a Streamlined Approach for Grading UML Diagrams“. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450381352. URL: https://doi.org/10.1145/3417990.3420052.

Feichas, Felipe A. und Rodrigo D. Seabra (2023). „Evaluation of Perception of Use of a Gamified Platform from the Student Perspective: An Approach for Studying Unified Modeling Language“. In: *Informatics in Education*. ISSN: 1648-5831. DOI: 10.15388/infedu.2023.22.

Fengler, Jörg (2017). *Feedback geben: Strategien und Übungen*. Julius Beltz.

Fernandes, Miguel Appleton und Vasco Amaral (2022). „A Simulation Framework for UML Education“. In: *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, S. 161–166. DOI: 10.1109/COMPSAC54236.2022.00031.

Foss, Sarah, Tatiana Urazova und Ramon Lawrence (2022). „Learning UML Database Design and Modeling with AutoER“. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '22. Montreal, Quebec, Canada: Association for Computing Machinery, S. 42–45. ISBN:

9781450394673. DOI: 10.1145/3550356.3559091. URL: https://doi.org/10.1145/3550356.3559091.

Gamma, Erich u. a. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional. ISBN: 0201633612.

Gauntlett, Nick (2007). „Literature review on formative assessment in higher education“. In: *Middlesex University*.

Gensheimer, Matthias, Florian Huber und Georg Hagel (2020). „Gamification in Software Engineering Education through Visual Novels“. In: *Proceedings of the 4th European Conference on Software Engineering Education*. ECSEE '20. Seeon/Bavaria, Germany: Association for Computing Machinery, S. 1–5. ISBN: 9781450377522. DOI: 10.1145/3396802.3396808. URL: https://doi.org/10.1145/3396802.3396808.

Ghezzi, Carlo und Dino Mandrioli (2006). „The Challenges of Software Engineering Education“. In: *Software Engineering Education in the Modern Age*. Hrsg. von Paola Inverardi und Mehdi Jazayeri. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 115–127. ISBN: 978-3-540-68204-2.

Gillioz, Anthony u. a. (2020). „Overview of the Transformer-based Models for NLP Tasks“. In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, S. 179–183. DOI: 10.15439/2020F20.

Goodfellow, Ian, Yoshua Bengio und Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Göpferich-Görnert, Susanne (2018). „9. Textverständlichkeit“. In: *Handbuch Text und Gespräch*. Hrsg. von Karin Birkner und Nina Janich. Berlin, Boston: De Gruyter, S. 229–248. ISBN: 9783110296051. DOI: doi:10.1515/9783110296051-009. URL: https://doi.org/10.1515/9783110296051-009.

Greff, Klaus u. a. (2017). „LSTM: A Search Space Odyssey“. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10, S. 2222–2232. DOI: 10.1109/TNNLS.2016.2582924.

Hannafin, Michael J. u. a. (2014). „Student-Centered, Open Learning Environments: Research, Theory, and Practice“. In: *Handbook of Research on Educational Communications and Technology*. Hrsg. von J. Michael Spector u. a. New York, NY: Springer New York, S. 641–651. ISBN: 978-1-4614-3185-5. DOI: 10.1007/978-1-4614-3185-5_51. URL: https://doi.org/10.1007/978-1-4614-3185-5_51.

Harris, Christopher G. und M. J. Stephens (1988). „A Combined Corner and Edge Detector“. In: *Alvey Vision Conference*, S. 147–151.

Hasker, Robert W. (Okt. 2011). „UMLGrader: An Automated Class Diagram Grader“. In: *J. Comput. Sci. Coll.* 27.1, S. 47–54. ISSN: 1937-4771.

Hasker, Robert W. und Mike Rowe (Juni 2011). „UMLint: Identifying Defects in UML Diagrams“. In: *2011 ASEE Annual Conference & Exposition*. 10.18260/1-2–18929. https://peer.asee.org/18929. Vancouver, BC: ASEE Conferences.

Hattie, John, Wolfgang Beywl und Klaus Zierer (2013). *Lernen sichtbar machen*. ger. Überarb. deutschsprachige Ausg. Hier auch später erschienene, unveränderte Nachdrucke ; Literaturverz. S. [309] - 376. - Hier auch später erschienene, unveränd. Nachdr. Baltmannsweiler: Schneider-Verl. Hohengehren, XXXVII, 439 S. ISBN: 978-3-8340-1190-9.

Hattie, John und Helen Timperley (2007). „The power of feedback“. In: *Review of educational research* 77.1, S. 81–112.

He, Kaiming u. a. (2016). „Deep Residual Learning for Image Recognition“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 770–778. DOI: 10.1109/CVPR.2016.90.

Hebb, Donald (1949). *The organization of behavior; a neuropsychological theory*. The organization of behavior; a neuropsychological theory. Oxford, England: Wiley, S. xix, 335–xix, 335.

Heer, Jeffrey und Maneesh Agrawala (2006). „Software Design Patterns for Information Visualization“. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5, S. 853–860. DOI: 10.1109/TVCG.2006.178.

Herout, Pavel und Premek Brada (2016). „UML-Test Application for Automated Validation of Students' UML Class Diagram“. In: *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEE&T)*, S. 222–226. DOI: 10.1109/CSEET.2016.33.

Hevner, Alan (Jan. 2007). „A Three Cycle View of Design Science Research“. In: *Scandinavian Journal of Information Systems* 19. Verfügbar unter: https://aisel.aisnet.org/sjis/vol19/iss2/4.

Hevner, Alan R. u. a. (März 2004). „Design Science in Information Systems Research“. In: *MIS Q.* 28.1, S. 75–105. ISSN: 0276-7783.

Hevner, Alan und Samir Chatterjee (2010). „Design Science Research in Information Systems“. In: *Design Research in Information Systems: Theo-*

ry and Practice. Boston, MA: Springer US, S. 9–22. ISBN: 978-1-4419-5653-8. DOI: 10.1007/978-1-4419-5653-8_2. URL: https://doi.org/10.1007/978-1-4419-5653-8_2.

Hinton, G. E. und R. R. Salakhutdinov (2006). „Reducing the Dimensionality of Data with Neural Networks“. In: *Science* 313.5786, S. 504–507. DOI: 10.1126/science.1127647. eprint: https://www.science.org/doi/pdf/10.1126/science.1127647. URL: https://www.science.org/doi/abs/10.1126/science.1127647.

Hochreiter, Sepp und Jürgen Schmidhuber (Nov. 1997). „Long Short-Term Memory“. In: *Neural Computation* 9.8, S. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Hochschulrektorenkonferenz (2015). „Standards und Leitlinien für die Qualitätssicherung im Europäischen Hochschulraum (ESG)“. In:

Huber, Florian und Georg Hagel (2020). „Work-in-Progress: Towards detection and syntactical analysis in UML class diagrams for software engineering education“. In: *2020 IEEE Global Engineering Education Conference (EDUCON)*, S. 3–7. DOI: 10.1109/EDUCON45650.2020.9125244.

— (2021). „The application of continuous practices in higher computer science education - A systematic literature review“. In: *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, S. 1618–1623. DOI: 10.23919/MIPRO52101.2021.9597101.

— (2022a). „Semi-automatic generation of textual exercises for software engineering education“. In: *2022 IEEE Global Engineering Education Conference (EDUCON)*, S. 51–56. DOI: 10.1109/EDUCON52537.2022.9766802.

— (2022b). „Tool-supported teaching of UML diagrams in software engineering education - A systematic literature review“. In: *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, S. 1404–1409. DOI: 10.23919/MIPRO55190.2022.9803560.

— (2022c). „Work-In-Progress: Converting textual software engineering class diagram exercises to UML models“. In: *2022 IEEE Global En-*

gineering Education Conference (EDUCON), S. 1–3. DOI: 10.1109/EDUCON52537.2022.9766593.

Huber, Florian u. a. (2023a). „From Difficulties to Functional Requirements - Deriving Requirements from Literature about Tool-Supported Teaching of UML Diagrams in Software Engineering Education“. In: *Proceedings of the 5th European Conference on Software Engineering Education*. ECSEE '23. Seeon/Bavaria, Germany: Association for Computing Machinery, S. 184–188. ISBN: 9781450399562. DOI: 10.1145/3593663.3593672. URL: https://doi.org/10.1145/3593663.3593672.

— (2023b). „Qualitative Requirements Elicitation of Student Requirements for Tool-Supported Teaching of UML Diagrams“. In: *Proceedings of the 5th European Conference on Software Engineering Education*. ECSEE '23. Seeon/Bavaria, Germany: Association for Computing Machinery, S. 189–193. ISBN: 9781450399562. DOI: 10.1145/3593663.3593673. URL: https://doi.org/10.1145/3593663.3593673.

Ichinohe, Yuta u. a. (2019). „Effectiveness of Automated Grading Tool Utilizing Similarity for Conceptual Modeling“. In: *Knowledge-Based Software Engineering: 2018*. Hrsg. von Maria Virvou, Fumihiro Kumeno und Konstantinos Oikonomou. Cham: Springer International Publishing, S. 117–126.

„IEEE Standard Glossary of Software Engineering Terminology“ (1990). In: *IEEE Std 610.12-1990*, S. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

Iqbal, Touseef und Shaima Qureshi (2020). „The survey: Text generation models in deep learning“. In: *Journal of King Saud University - Computer and Information Sciences*. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2020.04.001. URL: https://www.sciencedirect.com/science/article/pii/S1319157820303360.

„ISO/IEC/IEEE International Standard - Systems and software engineering – System life cycle processes“ (2015). In: *ISO/IEC/IEEE 15288 First edition 2015-05-15*, S. 1–118. DOI: 10.1109/IEEESTD.2015.7106435.

„ISO/IEC/IEEE International Standard - Systems and Software Engineering– Life Cycle Management– Part 2: Guidelines for the Application of ISO/IEC/IEEE 15288 (System Life Cycle Processes)“ (2018). In: *ISO/IEC/IEEE 24748-2:2018(E)*, S. 1–90. DOI: 10.1109/IEEESTD.2018.8764712.

Jang, Jungsun u. a. (Apr. 2020). „Narrative context-based data-to-text generation for ambient intelligence“. In: *Journal of Ambient Intelligence and Humanized Computing* 11.4, S. 1421–1429. ISSN: 1868-5145. DOI: 10.1007/s12652-019-01176-7. URL: https://doi.org/10.1007/s12652-019-01176-7.

Jocher, Glenn u. a. (Feb. 2022). *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Version v6.1. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936.

Jonassen, David (1999). „Designing constructivist learning environments“. In: *Instructional-Design Theories and Models: A New Paradigm of Instructional Theory*, S. 215–239.

Jurgelaitis, Mantas u. a. (2019). „Implementing gamification in a university-level UML modeling course: A case study“. In: *Computer Applications in Engineering Education* 27.2, S. 332–343. DOI: https://doi.org/10.1002/cae.22077. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cae.22077. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22077.

Jütte, Wolfgang, Markus Walber und Claudia Lobe (2017). „Innovativer Lehre auf der Spur: Beobachtungsperspektiven und Forschungszugriffe“. In: *Das Neue in der Hochschullehre: Lehrinnovationen aus der Perspektive der hochschulbezogenen Lehr-Lern-Forschung*. Wiesbaden: Springer Fachmedien Wiesbaden, S. 1–15. ISBN: 978-3-658-13777-9. DOI: 10.1007/978-3-658-13777-9_1. URL: https://doi.org/10.1007/978-3-658-13777-9_1.

Kember, David (1997). „A reconceptualisation of the research into university academics' conceptions of teaching“. In: *Learning and Instruction* 7.3, S. 255–275. ISSN: 0959-4752. DOI: https://doi.org/10.1016/S0959-4752(96)00028-X. URL: https://www.sciencedirect.com/science/article/pii/S095947529600028X.

Khurana, Diksha u. a. (Jan. 2023). „Natural language processing: state of the art, current trends and challenges“. In: *Multimedia Tools and Applications* 82.3, S. 3713–3744. ISSN: 1573-7721. DOI: 10.1007/s11042-022-13428-4. URL: https://doi.org/10.1007/s11042-022-13428-4.

Kibble, Jonathan D. (2017). „Best practices in summative assessment“. In: *Advances in Physiology Education* 41.1. PMID: 28188198, S. 110–119. DOI: 10.1152/advan.00116.2016. eprint: https://doi.org/10.1152/advan.00116.2016. URL: https://doi.org/10.1152/advan.00116.2016.

Kitchenham, Barbara (2004). *Procedure for undertaking systematic reviews*. Techn. Ber. Joint Technical Report. Computer Science Department, Keele University (TRISE-0401) und National ICT Australia Ltd (0400011T.1). URL: http://www.it.hiof.no/~haraldh/misc/2016-08-22-smat/Kitchenham-Systematic-Review-2004.pdf.

Kitchenham, Barbara und Stuart Charters (2007). *Guidelines for performing systematic literature reviews in software engineering*. Techn. Ber. Keele University und University of Durham: EBSE Technical Report EBSE-2007-01. URL: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

Kleuker, Stephan (2018). „Optimierung des Designmodells“. In: *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*. Wiesbaden: Springer Fachmedien Wiesbaden, S. 179–227. ISBN: 978-3-658-19969-2. DOI: 10.1007/978-3-658-19969-2_8. URL: https://doi.org/10.1007/978-3-658-19969-2_8.

Knobloch, Jan, Jonas Kaltenbach und Bernd Bruegge (2018). „Increasing Student Engagement in Higher Education Using a Context-Aware Q&A Teaching Framework“. In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. ICSE-SEET '18. Gothenburg, Sweden: Association for Computing Machinery, S. 136–145. ISBN: 9781450356602. DOI: 10.1145/3183377.3183389. URL: https://doi.org/10.1145/3183377.3183389.

Krüger, Ralph (2021). „Die Transformer-Architektur für Systeme zur neuronalen maschinellen Übersetzung -eine popularisierende Darstellung“. In: S. 278–324. ISSN: 1867-4844.

Krusche, Stephan u. a. (2018). „50 Years of Software Engineering: Challenges, Results, and Opportunities in Its Education“. In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018. Larnaca, Cyprus: Association

for Computing Machinery, S. 362–363. ISBN: 9781450357074. DOI: 10.1145/3197091.3205848. URL: https://doi.org/10.1145/3197091.3205848.

Krusche, Stephan u. a. (2020). „An interactive learning method to engage students in modeling“. In: Seoul, South Korea: Association for Computing Machinery, S. 12–22. ISBN: 9781450371247. DOI: 10.1145/3377814.3381701. URL: https://doi.org/10.1145/3377814.3381701.

Kruus, Helena, Tarmo Robal und Gert Jervan (2014). „Teaching modeling in SysML/UML and problems encountered“. In: *2014 25th EAEEIE Annual Conference (EAEEIE)*, S. 33–36. DOI: 10.1109/EAEEIE.2014.6879380.

Kurkovsky, Stan (2015). „Teaching Software Engineering with LEGO Serious Play“. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '15. Vilnius, Lithuania: Association for Computing Machinery, S. 213–218. ISBN: 9781450334402. DOI: 10.1145/2729094.2742604. URL: https://doi.org/10.1145/2729094.2742604.

LeCun, Yann und Françoise Fogelman-Soulié (1987). „Modèles connexionnistes de l'apprentissage“. In: *Intellectica* 2.1, S. 114–143.

Lee, Eunbae und Michael J. Hannafin (Aug. 2016). „A design framework for enhancing engagement in student-centered learning: own it, learn it, and share it“. In: *Educational Technology Research and Development* 64.4, S. 707–734. ISSN: 1556-6501. DOI: 10.1007/s11423-015-9422-5. URL: https://doi.org/10.1007/s11423-015-9422-5.

Lee, Roger Y. (2013). *Software Engineering: A Hands-On Approach*. Atlantis Press Paris. ISBN: 978-94-6239-006-5. DOI: 10.2991/978-94-6239-006-5. URL: https://doi.org/10.2991/978-94-6239-006-5.

Lethbridge, Timothy C. u. a. (2011). „Teaching UML using umple: Applying model-oriented programming in the classroom“. In: *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, S. 421–428. DOI: 10.1109/CSEET.2011.5876118.

Lewis, James R. (2018). „The System Usability Scale: Past, Present, and Future“. In: *International Journal of Human–Computer Interaction* 34.7, S. 577–590. DOI: 10.1080/10447318.2018.1455307. eprint: https://doi.org/10.1080/10447318.2018.1455307. URL: https://doi.org/10.1080/10447318.2018.1455307.

Lin, Tianyang u. a. (2021). *A Survey of Transformers*. DOI: 10.48550/ARXIV.2106.04554. URL: https://arxiv.org/abs/2106.04554.

Lipnevich, Anastasiya A. und Ernesto Panadero (Dez. 2021). „A Review of Feedback Models and Theories: Descriptions, Definitions, and Conclusions“. In: *Frontiers in Education* 6. DOI: 10.3389/feduc.2021.720195. URL: http://dx.doi.org/10.3389/feduc.2021.720195.

Liskov, Barbara (1987). „Keynote Address - Data Abstraction and Hierarchy“. In: *Addendum to the Proceedings on Object-Oriented Programming Systems, Languages and Applications (Addendum)*. OOPSLA '87. Orlando, Florida, USA: Association for Computing Machinery, S. 17–34. ISBN: 0897912667. DOI: 10.1145/62138.62141. URL: https://doi.org/10.1145/62138.62141.

Liskov, Barbara H. und Jeannette M. Wing (Nov. 1994). „A Behavioral Notion of Subtyping“. In: *ACM Trans. Program. Lang. Syst.* 16.6, S. 1811–1841. ISSN: 0164-0925. DOI: 10.1145/197320.197383. URL: https://doi.org/10.1145/197320.197383.

Liu, Shu u. a. (2018). „Path Aggregation Network for Instance Segmentation“. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, S. 8759–8768. DOI: 10.1109/CVPR.2018.00913.

Liu, Yiheng u. a. (2023). *Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models*. arXiv: 2304.01852 [cs.CL].

Manaswi, Navin Kumar (2018). „Deep Learning with Applications Using Python : Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras. Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras“. In: Berkeley, CA: Apress. ISBN: 978-1-4842-3516-4. DOI: 10.1007/978-1-4842-3516-4. URL: https://doi.org/10.1007/978-1-4842-3516-4.

Mandl, Heinz und Helmut F Friedrich (1992). „Lern-und Denkstrategien“. In: *Analyse und Intervention. Göttingen: Hogrefe*.

Martin, Robert (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson Education, Inc. ISBN: 978-0-13-449416-6.

Mayring, Philipp (2008). *Qualitative Inhaltsanalyse. Grundlagen und Techniken*. de. Beltz Pädagogik. Weinheim, Germany: Julius Beltz.

Mayring, Philipp (2010). „Qualitative Inhaltsanalyse“. In: *Handbuch Qualitative Forschung in der Psychologie*. Hrsg. von Günter Mey und Katja Mruck. Wiesbaden: VS Verlag für Sozialwissenschaften, S. 601–613. ISBN: 978-3-531-92052-8. DOI: 10.1007/978-3-531-92052-8_42. URL: https://doi.org/10.1007/978-3-531-92052-8_42.

Mayring, Philipp und Eva Brunner (2006). „Qualitative Textanalyse - Qualitative Inhaltsanalyse“. In: *Von der Idee zur Forschungsarbeit. Forschen in Sozialarbeit und Sozialwissenschaft*, S. 453–462.

Mayring, Philipp und Thomas Fenzl (2022). „Qualitative Inhaltsanalyse“. In: *Handbuch Methoden der empirischen Sozialforschung*. Hrsg. von Nina Baur und Jörg Blasius. Wiesbaden: Springer Fachmedien Wiesbaden, S. 691–706. ISBN: 978-3-658-37985-8. DOI: 10.1007/978-3-658-37985-8_43. URL: https://doi.org/10.1007/978-3-658-37985-8_43.

McClelland, James L., David E. Rumelhart und Geoffrey E. Hinton (1986). „The Appeal of Parallel Distributed Processing“. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, S. 3–44. ISBN: 026268053X.

McCulloch, Warren und Walter Pitts (Dez. 1943). „A logical calculus of the ideas immanent in nervous activity“. In: *The bulletin of mathematical biophysics* 5.4, S. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

Mishra, Alok, Nergiz Ercil Cagiltay und Ozkan Kilic (2007). „Software engineering education: some important dimensions“. In: *European Journal of Engineering Education* 32.3, S. 349–361. DOI: 10.1080/03043790701278607. eprint: https://doi.org/10.1080/03043790701278607. URL: https://doi.org/10.1080/03043790701278607.

Modi, Astha u. a. (Okt. 2022). „Sentiment Analysis of Twitter Feeds Using Flask Environment: A Superior Application of Data Analysis“. In: *Annals of Data Science*. ISSN: 2198-5812. DOI: 10.1007/s40745-022-00445-1. URL: https://doi.org/10.1007/s40745-022-00445-1.

Modi, Salisu, Hanan Abdulrahman Taher und Hoger Mahmud (Juni 2021). „A Tool to Automate Student UML diagram Evaluation“. In: *Academic Journal of Nawroz University* 10.2, S. 189–198. DOI: 10.25007/ajnu.

v10n2a1035. URL: https://journals.nawroz.edu.krd/index.php/ajnu/article/view/1035.

Montesinos López, Osval Antonio, Abelardo Montesinos López und Jose Crossa (2022). „Convolutional Neural Networks“. In: *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. Cham: Springer International Publishing, S. 533–577. ISBN: 978-3-030-89010-0. DOI: 10.1007/978-3-030-89010-0_13. URL: https://doi.org/10.1007/978-3-030-89010-0_13.

Müller-Amthor, Martina u. a. (2020). „Scrum Higher Education – The Scrum Master Supports as Solution-focused Coach“. In: *2020 IEEE Global Engineering Education Conference (EDUCON)*, S. 948–952. DOI: 10.1109/EDUCON45650.2020.9125304.

Nepal, Upesh und Hossein Eslamiat (Jan. 2022). „Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs“. In: *Sensors* 22.2. ISSN: 1424-8220. DOI: 10.3390/s22020464. URL: http://dx.doi.org/10.3390/s22020464.

Neudecker, Angelika (2021). *Digitale Transformation und John Hatties Bildungsforschung: Implikationen auf den Bildungsfaktor Feedback*. ger. DOI: https://doi.org/10.25656/01:23653.

Nussbaumer, Markus (1991). *Ansätze zu einer sprachwissenschaftlichen Begründung eines Kriterienrasters zur Beurteilung von schriftlichen Schülertexten*. Berlin, New York: Max Niemeyer Verlag. ISBN: 9783111632308. DOI: doi:10.1515/9783111632308. URL: https://doi.org/10.1515/9783111632308.

O'Neill, Geraldine und Tim Mcmahon (Jan. 2005). „Student-centred learning: What does it mean for students and lecturers?“ In: *Emerging Issues in the Practice of University Learning and Teaching* 1.

Object Management Group (2015). *About the XML Metadata Interchange Specification Version 2.5.1*. https://www.omg.org/spec/XMI/2.5.1. [Accessed 15-10-2023].

Oestereich, Bernd und Stefan Bremer (2009). *Die UML-Kurzreferenz 2.3 für die Praxis*. Oldenbourg Wissenschaftsverlag.

Oguz, Damla und Kaya Oguz (2019). „Perspectives on the Gap Between the Software Industry and the Software Engineering Education“. In: *IEEE Access* 7, S. 117527–117543. DOI: 10.1109/ACCESS.2019.2936660.

OpenAI (2023). *Introducing ChatGPT — openai.com*. https://openai.com/blog/chatgpt. [Accessed 25-May-2023].

Ortner, Erich (Feb. 2002). „SprachingenieurwesenEmpfehlung zur inhaltlichen Weiterentwicklung der (Wirtschafts-)Informatik“. In: *Informatik-Spektrum* 25.1, S. 39–51. ISSN: 1432-122X. DOI: 10.1007/s002870100203. URL: https://doi.org/10.1007/s002870100203.

Ouhbi, Sofia und Nuno Pombo (2020). „Software Engineering Education: Challenges and Perspectives“. In: *2020 IEEE Global Engineering Education Conference (EDUCON)*, S. 202–209. DOI: 10.1109/EDUCON45650.2020.9125353.

Ouyang, Long u. a. (2022). *Training language models to follow instructions with human feedback*. arXiv: 2203.02155 [cs.CL].

Partalidou, Eleni u. a. (2019). „Design and implementation of an open source Greek POS Tagger and Entity Recognizer using spaCy“. In: *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, S. 337–341.

Pattanayak, Santanu (2017). „Introduction to Deep-Learning Concepts and TensorFlow“. In: *Pro Deep Learning with TensorFlow: A Mathematical Approach to Advanced Artificial Intelligence in Python*. Berkeley, CA: Apress, S. 89–152. ISBN: 978-1-4842-3096-1. DOI: 10.1007/978-1-4842-3096-1_2. URL: https://doi.org/10.1007/978-1-4842-3096-1_2.

Plebe, Alessio und Giorgio Grasso (Dez. 2019). „The Unbearable Shallow Understanding of Deep Learning“. In: *Minds and Machines* 29.4, S. 515–553. ISSN: 1572-8641. DOI: 10.1007/s11023-019-09512-8. URL: https://doi.org/10.1007/s11023-019-09512-8.

Porst, Rolf (2014). *Fragebogen: Ein Arbeitsbuch*. Wiesbaden: Springer Fachmedien Wiesbaden. ISBN: 978-3-658-02118-4. DOI: 10.1007/978-3-658-02118-4. URL: https://doi.org/10.1007/978-3-658-02118-4.

Py, Dominique, Ludovic Auxepaules und Mathilde Alonso (Okt. 2013). „Diagram, a Learning Environment for Initiation to Object-Oriented Modeling with UML Class Diagrams“. In: *Journal of Interactive Learning Research* 24.4, S. 425–446. ISSN: 1093-023X. URL: https://www.learntechlib.org/p/41241.

R.M. Harden, Joy Crosby (2000). „AMEE Guide No 20: The good teacher is more than a lecturer - the twelve roles of the teacher“. In: *Medi-*

cal Teacher 22.4, S. 334–347. DOI: 10.1080/014215900409429. eprint: https://doi.org/10.1080/014215900409429. URL: https://doi.org/10.1080/014215900409429.

Radford, Alec u. a. (2019). „Language models are unsupervised multitask learners“. In: *OpenAI blog* 1.8, S. 9.

Ramírez-Noriega, Alan u. a. (2020). „A software tool to generate a Model-View-Controller architecture based on the Entity-Relationship Model“. In: *2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT)*, S. 57–63. DOI: 10.1109/CONISOFT50191.2020.00018.

Redmon, Joseph und Ali Farhadi (2016). *YOLO9000: Better, Faster, Stronger*. arXiv: 1612.08242 [cs.CV].

— (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767 [cs.CV].

Redmon, Joseph u. a. (2016). „You Only Look Once: Unified, Real-Time Object Detection“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 779–788. DOI: 10.1109/CVPR.2016.91.

Reischmann, Tobias und Herbert Kuchen (2016). „Towards an E-Assessment Tool for Advanced Software Engineering Skills“. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling '16. Koli, Finland: Association for Computing Machinery, S. 81–90. ISBN: 9781450347709. DOI: 10.1145/2999541.2999550. URL: https://doi.org/10.1145/2999541.2999550.

Reuter, Rebecca u. a. (2019). „Using Augmented Reality in Software Engineering Education? First insights to a comparative study of 2D and AR UML modeling“. In: *Proceedings of the 52nd Hawaii International Conference on System Sciences 2019*.

Reuter, Rebecca u. a. (2020). „Insights in Students' Problems during UML Modeling“. In: *2020 IEEE Global Engineering Education Conference (EDUCON)*, S. 592–600. DOI: 10.1109/EDUCON45650.2020.9125110.

Richardson, Leonard und Sam Ruby (2007). *RESTful web services*. O'Reilly Media, Inc. ISBN: 978-0-596-52926-0.

Rodrigues, Claudia Susie C., Cláudia M. L. Werner und Luiz Landau (2016). „VisAr3D: An Innovative 3D Visualization of UML Models“. In: *2016 IE-*

EE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), S. 451–460. ISBN: 978-1-4503-4205-6.

Rosenblatt, Frank, Alexander Stieber und Robert H. Schatz (Jan. 1957). *The Perceptron - A Perceiving and Recognizing Automaton*. Buffalo, N.Y.: Cornell Aeronautical Laboratory Inc. URL: https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf.

Rowley, Jennifer und Frances Slack (Juni 2004). „Conducting a literature review“. en. In: *Management Research News* 27.6, S. 31–39. ISSN: 0140-9174. DOI: https://doi.org/10.1108/01409170410784185.

Rukmono, Satrio Adi und Michel R. V. Chaudron (2022). „Guiding Peer-Feedback in Learning Software Design Using UML“. In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*. ICSE-SEET '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, S. 122–133. ISBN: 9781450392259. DOI: 10.1145/3510456.3514148. URL: https://doi.org/10.1145/3510456.3514148.

Rumelhart, David E., Geoffrey E. Hinton und Ronald J. Williams (Okt. 1986a). „Learning representations by back-propagating errors“. In: *Nature* 323.6088, S. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

Rumelhart, David E., James L. McClelland und PDP Research Group (Juli 1986b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press. ISBN: 9780262291408. DOI: 10.7551/mitpress/5236.001.0001. URL: https://doi.org/10.7551/mitpress/5236.001.0001.

Rupp, Chris, Stefan Queins und SOPHISTen (2014). *Requirements-Engineering und -Management. Aus der Praxis von klassisch bis agil*. Carl Hanser Verlag GmbH & Co. KG, S. 570. ISBN: 978-3-446-43893-4.

Sadler, D Royce (Juni 1989). „Formative assessment and the design of instructional systems“. In: *Instructional Science* 18.2, S. 119–144.

Saini, Rijul u. a. (2019). „Teaching Modelling Literacy: An Artificial Intelligence Approach“. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, S. 714–719. DOI: 10.1109/MODELS-C.2019.00108.

Schots, Marcelo u. a. (2010). „A Study on the Application of the PREViA Approach in Modeling Education“. In: *2010 XXIX International Conference of the Chilean Computer Science Society*, S. 96–101. DOI: 10.1109/SCCC.2010.29.

Schriver, Karen A. (1989). „Evaluating text quality: the continuum from text-focused to reader-focused methods“. In: *IEEE Transactions on Professional Communication* 32.4, S. 238–255. DOI: 10.1109/47.44536.

Sedrakyan, Gayane und Monique Snoeck (2012). „Technology-Enhanced Support for Learning Conceptual Modeling“. In: *Enterprise, Business-Process and Information Systems Modeling*. Hrsg. von Ilia Bider u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 435–449. ISBN: 978-3-642-31072-0.

Seidl, Martina u. a. (2015). *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Cham: Springer International Publishing. ISBN: 978-3-319-12742-2. DOI: 10.1007/978-3-319-12742-2. URL: https://doi.org/10.1007/978-3-319-12742-2.

Shneiderman, Ben u. a. (2017). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 6th. Pearson. ISBN: 978-1-292-15392-6.

Shute, Valerie J (2008). „Focus on formative feedback“. In: *Review of educational research* 78.1, S. 153–189.

Sieber, Peter, Hrsg. (1994). *Sprachfähigkeiten - besser als ihr Ruf und nötiger denn je! Ergebnisse und Folgerungen aus einem Forschungsprojekt.* Aarau u.a.: Sauerländer, 383 S. ISBN: 3-7941-3770-1.

Singh, Prabhsimran, Younes Boubekeur und Gunter Mussbacher (2022). „Detecting Mistakes in a Domain Model“. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '22. Montreal, Quebec, Canada: Association for Computing Machinery, S. 257–266. ISBN: 9781450394673. DOI: 10.1145/3550356.3561583. URL: https://doi.org/10.1145/3550356.3561583.

Soler, Josep u. a. (2010). „A web-based e-learning tool for UML class diagrams“. In: *IEEE EDUCON 2010 Conference*, S. 973–979. DOI: 10.1109/EDUCON.2010.5492473.

spaCy 101: Everything you need to know (2022). https://spacy.io/usage/spacy-101. Accessed: 2022-08-16.

Starke, Gernot (2014). *Effektive Softwarearchitekturen. Ein praktischer Leitfaden*. Munich, Germany: Carl Hanser Verlag GmbH Co KG. ISBN: 978-3-446-43614-5.

Stikkolorum, D. R., F. Gomes de Oliveira Neto und M. R. V. Chaudron (2018). „Evaluating Didactic Approaches Used by Teaching Assistants for Software Analysis and Design Using UML“. In: *Proceedings of the 3rd European Conference of Software Engineering Education*. ECSEE'18. Seeon/ Bavaria, Germany: Association for Computing Machinery, S. 122–131. ISBN: 9781450363839. DOI: 10.1145/3209087.3209107. URL: https://doi.org/10.1145/3209087.3209107.

Stikkolorum, Dave, Truong Ho-Quang und Michel Chaudron (2015). „Revealing Students' UML Class Diagram Modelling Strategies with WebUML and LogViz“. In: *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, S. 275–279. DOI: 10.1109/SEAA.2015.77.

Stikkolorum, Dave u. a. (2019). „Towards Automated Grading of UML Class Diagrams with Machine Learning“. In: *BNAIC/BENELEARN*.

Stuurman, Sylvia, Harrie Passier und Erik Barendsen (2016). „Analyzing Students' Software Redesign Strategies“. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. Koli Calling '16. Koli, Finland: Association for Computing Machinery, S. 110–119. ISBN: 9781450347709. DOI: 10.1145/2999541.2999559. URL: https://doi.org/10.1145/2999541.2999559.

Thomas, David und Andrew Hunt (2019). *The pragmatic programmer*. Addison-Wesley Professional. ISBN: 9780135957059.

Thomasson, Benjy, Mark Ratcliffe und Lynda Thomas (Juni 2006). „Identifying Novice Difficulties in Object Oriented Design“. In: *SIGCSE Bull.* 38.3, S. 28–32. ISSN: 0097-8418. DOI: 10.1145/1140123.1140135. URL: https://doi.org/10.1145/1140123.1140135.

Tsichritzis, Dennis (1997). „The Dynamics of Innovation“. In: *Beyond Calculation: The Next Fifty Years of Computing*. New York, NY: Springer New York, S. 259–265. ISBN: 978-1-4612-0685-9. DOI: 10.1007/978-1-

4612-0685-9_19. URL: https://doi.org/10.1007/978-1-4612-0685-9_19.

Vaswani, Ashish u. a. (2017). *Attention Is All You Need*. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

Vogel, Oliver u. a. (2011). „Software Architecture: A Comprehensive Framework and Guide for Practitioners“. In: Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-19736-9. DOI: 10.1007/978-3-642-19736-9. URL: https://doi.org/10.1007/978-3-642-19736-9.

Vyshnevskyi, Oleksandr S (2020). „Impact of digitalization on industry: problems of definition in EU countries“. In: *Economy of industry* 1 (89), S. 31–44.

Walde, Janette F. (2005). „Das Mehrschichtige Perzeptron (MLP)“. In: *Design Künstlicher Neuronaler Netze: Ein Leitfaden zur effizienten Handhabung mehrschichtiger Perzeptrone*. Wiesbaden: Deutscher Universitätsverlag, S. 9–31. ISBN: 978-3-322-81211-7. DOI: 10.1007/978-3-322-81211-7_2. URL: https://doi.org/10.1007/978-3-322-81211-7_2.

Wang, Chien-Yao u. a. (2020). „CSPNet: A New Backbone that can Enhance Learning Capability of CNN“. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, S. 1571–1580. DOI: 10.1109/CVPRW50498.2020.00203.

Wang, Haohan, Bhiksha Raj und Eric P. Xing (2017). „On the Origin of Deep Learning“. In: *CoRR* abs/1702.07800. arXiv: 1702.07800. URL: http://arxiv.org/abs/1702.07800.

Wei, Bingyang u. a. (2016). „A Conceptual Graphs Framework for Teaching UML Model-Based Requirements Acquisition“. In: *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEE&T)*, S. 71–75. DOI: 10.1109/CSEET.2016.35.

Weinstein, Claire E und Richard E Mayer (1983). „The teaching of learning strategies“. In: *Innovation Abstracts*. Bd. 5. 32. ERIC, n32.

Wilde, Thomas und Thomas Hess (Aug. 2007). „Forschungsmethoden der Wirtschaftsinformatik“. In: *WIRTSCHAFTSINFORMATIK* 49.4, S. 280–287. ISSN: 1861-8936. DOI: 10.1007/s11576-007-0064-z. URL: https://doi.org/10.1007/s11576-007-0064-z.

Winne, Philip H und Deborah L Butler (1994). „Student cognition in learning from teaching“. In: *International encyclopedia of education* 2, S. 5738–5775.

Yang, Jeong, Youlg Lee und Kai H. Chang (2017). „Initial Evaluation of JaguarCode: A Web-Based Object-Oriented Programming Environment with Static and Dynamic Visualization“. In: *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, S. 152–161. DOI: 10.1109/CSEET.2017.32.

Yang, Jeong u. a. (2015). „Enhancing object-oriented programming education using static and dynamic visualization“. In: *2015 IEEE Frontiers in Education Conference (FIE)*, S. 1–5. DOI: 10.1109/FIE.2015.7344152.

Yigitbas, Enes u. a. (2022). „Gamification-Based UML Learning Environment in Virtual Reality“. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '22. Montreal, Quebec, Canada: Association for Computing Machinery, S. 27–31. ISBN: 9781450394673. DOI: 10.1145/3550356.3559088. URL: https://doi.org/10.1145/3550356.3559088.

Young, Tom u. a. (2018). „Recent Trends in Deep Learning Based Natural Language Processing [Review Article]“. In: *IEEE Computational Intelligence Magazine* 13.3, S. 55–75. DOI: 10.1109/MCI.2018.2840738.

Zhou, Xinyu u. a. (2017). „EAST: An Efficient and Accurate Scene Text Detector“. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 2642–2651. DOI: 10.1109/CVPR.2017.283.