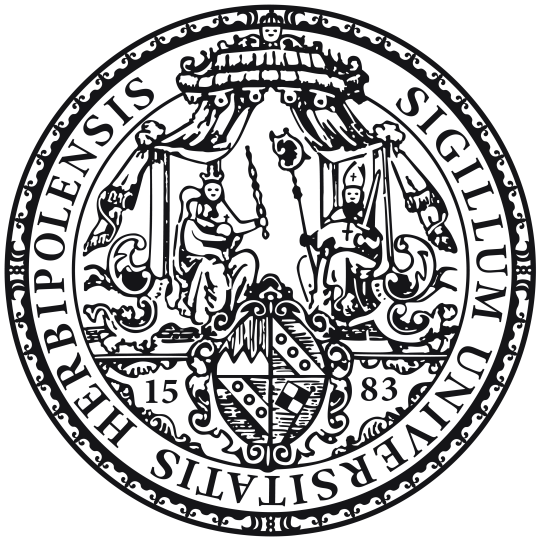
Feedback-Generierung für offene, strukturierte Aufgaben in E-Learning-Systemen

vorgelegt von

Marianus Ifland

Würzburg, 2014



Dissertation zur Erlangung des naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität Würzburg

Danksagung

In erster Linie möchte ich meinem Doktorvater Frank Puppe herzlichen Dank aussprechen, der sich stets Zeit für mich und meine wissenschaftlichen Tätigkeiten genommen und dafür auch an Wochenenden und Feiertagen Stunden geopfert hat.

Weiterhin danke ich all meinen Kollegen und Studierenden, die mich über die vergangenen Jahre begleitet, mir eine angenehme Arbeitsumgebung ermöglicht und mich bei meinen Projekten und verschiedensten anderen Problemen unterstützt haben: Alexander Dallmann, Alexander Hörnlein, Andreas Hotho, Astrid Weingart, Beate Navarro Bullock, Daniel Zoller, Edgar, Felix Herrmann, Florian Lemmerich, Georg Dietrich, Georg Fette, Gustavo Kuhn Andriotti, Joachim Baumeister, Jochen Reutelshöfer, Jürgen Helmerich, Kirsten Pearson, Martin Atzmüller, Martin Becker, Martin Töpfer, Martina Freiberg, Max Ertl, Peter Klügl, Petra Braun, Philip-Daniel Beck, Reinhard Hatko, Thomas Niebler, Christian Schneider, Julian Ott, Jürgen Zöller, und Michael Jedich.

Ich widme diese Arbeit meinen Familien, die mir vor allem in schwierigen Phasen den Rücken frei gehalten haben, so dass ich diese Arbeit mit klarem Kopf und Konzentration abschließen konnte. Auch ihnen danke ich herzlichst, im Besonderen meiner Frau Katharina, die immer da ist.

Würzburg, August 2014

Inhaltsverzeichnis

1. Einführung

1.1. Motivation

„Lehre bildet Geister; doch Übung macht den Meister."

Dieses bekannte deutsche Sprichwort besagt, dass vor allem das Einüben, also die tatsächliche Ausführung einer Tätigkeit, zu einer Verbesserung der Leistungsfähigkeit führt. Im Kontext der Ausbildung an Schulen und Hochschulen bedeutet dies, dass es wichtig ist, Schülern und Studierenden ausreichend viele Übungsmöglichkeiten anzubieten. Die Rückmeldung über die ausgeführte Tätigkeit bzw. deren Ergebnis ist wichtig, um einen Lerneffekt zu erzielen. Die von Lehrpersonal bei einer „Korrektur" erstellte Rückmeldung, auch Feedback genannt, ist jedoch teuer, da der zeitliche Aufwand je nach Art der Aufgabe beträchtlich ist. Das Ergebnis muss untersucht, Fehler und eine korrekte Lösung müssen aufgezeigt werden. Auch die zeitliche Verzögerung zwischen Ausführung der Tätigkeit und Rückmeldung ist bei der Korrektur schriftlicher Aufgaben ein Nachteil.

Eine Lösung dieser Problematik stellen E-Learning-Systeme dar. Geeignete Systeme können nicht nur Lernstoff präsentieren, sondern auch Übungsaufgaben anbieten und nach deren Bearbeitung quasi unmittelbar entsprechendes Feedback generieren. Zwar ist die technische Implementierung und die Aufbereitung und Eingabe von Lerninhalten aufwändig, doch kann ein laufendes System ohne zusätzlichen Aufwand von nahezu beliebig vielen Lernenden verwendet werden. Durch die rasante Verbreitung des Internets in den letzten 25 Jahren, die dazu führte, dass (zumindest in den Industrienationen) nahezu jede Person über entsprechende kostengünstige Zugangsmöglichkeiten verfügt, haben gerade webbasierte Trainings- und Tutorsysteme an Attraktivität gewonnen. Sie können ohne Aufwand auf Seiten der Benutzer technisch und inhaltlich stets aktuell gehalten werden. Zudem sind sie bei Verwendung entsprechender Technologien auf allen Endgeräten mit gängigen Plattformen lauffähig.

Es ist jedoch im Allgemeinen nicht einfach, maschinelle Verfahren zu implementieren, die Bearbeitungen von Übungsaufgaben korrigieren und entsprechendes Feedback erstellen. Für einige Aufgabentypen, wie beispielsweise Multiple-Choice-Aufgaben, ist dies zwar trivial, doch sind diese vor allem dazu gut geeignet, sogenanntes Faktenwissen abzuprüfen. Das Einüben von Lernzielen im Bereich der Anwendung ist damit kaum möglich. Die Behandlung dieser nach gängigen Taxonomien höheren kognitiven Lernziele erlauben sogenannte offene Aufgabentypen, deren Bearbeitung meist durch die Erstellung eines Freitexts in natürlicher Sprache erfolgt. Die Information bzw. das Wissen, das Lernende eingeben, liegt hier also in sogenannter „unstrukturierter" Form vor. Dieses

unstrukturierte Wissen ist maschinell nur schwer verwertbar, sodass sich Trainingssysteme, die Aufgaben dieser Art stellen und entsprechende Rückmeldung geben, bisher nicht durchgesetzt haben.

Es existieren jedoch auch offene Aufgabentypen, bei denen Lernende das Wissen in strukturierter Form eingeben, so dass es maschinell leichter zu verwerten ist. Für Aufgaben dieser Art lassen sich somit Trainingssysteme erstellen, die eine gute Möglichkeit darstellen, Schülern und Studierenden auch für praxisnahe Anwendungen viele Übungsmöglichkeiten zur Verfügung zu stellen, ohne das Lehrpersonal zusätzlich zu belasten.

1.2. Aufbau der Arbeit

In den unmittelbar folgenden Abschnitten wird eine kurze Einführung in die Thematik von E-Learning, Trainings- und Tutorsystemen gegeben. Anschließend erfolgt eine Erläuterung verschiedener Kontexte, in welchen Aufgaben an Hochschulen gestellt werden und welche Arten von Feedback dabei gegeben werden können. Es folgt die Beschreibung einer Ergänzung der aus der Literatur bekannten Typologien von Aufgaben. Aus dieser Ergänzung lässt sich der Typ der offenen, strukturierten Aufgabe mit zwei Untertypen ableiteten. Für diese Untertypen werden Verfahren skizziert, mit welchen für entsprechende Bearbeitungen dieser Aufgaben Feedback generiert werden kann. Der Fokus dieser Arbeit liegt auf den Erfahrungen mit dem praktischen Einsatz von vier Trainingssystemen bzw. deren Komponenten, die im Rahmen dieser Arbeit entwickelt wurden, und deren Konzeption, Implementierung und Evaluation in den Kapiteln 2 bis 5 beschrieben wird. Da diese Trainingssysteme in einer Universität Würzburg eingesetzt wurden, beziehen sich die im Folgenden beschrieben Konzepte auf den Kontext der Ausbildung an Hochschulen. Sie sind allerdings grundsätzlich auch auf schulische und betriebliche Aus- und Weiterbildung anwendbar. In Kapitel 6 wird die Arbeit zusammengefasst und ein Ausblick gegeben.

1.3. Einführung in E-Learning

Eine allgemein anerkannte, klare Definition des „E-Learning“, also des elektronischen Lernens, gibt es bisher nicht. Es umfasst in einer sehr allgemeinen Definition alle Vorgänge, die Lehr- und Lernprozesse in irgendeiner Weise durch elektronische bzw. digitale Informations- und Kommunikationstechnologien unterstützten. Dies umfasst beispielsweise die Präsentation von Lerninhalten, die Unterstützung der Kommunikation von beteiligten Personen, die Unterstützung der Verwaltung von Lernprozessen, aber auch das Stellen von Übungsaufgaben durch elektronische Systeme mit entsprechender Feedback-Generierung. In diesem Zusammenhang wird auch oft der Begriff „E-Assessment“ verwendet. Dabei steht allerdings meist die Messung der Leistungsfähigkeit der Probanden im Mittelpunkt, nicht aber ein möglicher Lerneffekt, der durch die Bearbeitung der Aufgaben entstehen kann.

An vielen Hochschulen hat sich E-Learning bis heute vor allem in Form von „Blended Learning“ etabliert. Dies bedeutet, dass E-Learning-Angebote nicht für sich allein stehen, sondern mit traditionellen Lehrangeboten vermischt (engl. *blended*) werden. Üblich ist die Vermittlung von Grundlagen in der Präsenzlehre, also zum Beispiel durch Frontalunterricht in klassischen Vorlesungen. Die elektronische Bereitstellung von Vorlesungsmaterialien ist mittlerweile weit verbreitet. Zum Einüben des Lernstoffs können Trainings- und (intelligente) Tutorsysteme verwendet werden. Darunter werden elektronische Systeme verstanden, die den Lernenden Informationen präsentieren, sie anleiten, entsprechende Aktionen ausführen lassen und entsprechendes Feedback geben. Die Intelligenz dieser Tutorsysteme bezieht sich auf die Tatsache, dass die Rückmeldungen auf die Lernenden individuell angepasst sind, was u.a. ein internes Modell des Lernenden erfordert.

Bei den in dieser Arbeit vorgestellten Systemen liegt der Fokus jedoch nicht in der Vermittlung von Wissen, sondern in der tatsächlichen, praxisnahen Ausführung von Tätigkeiten. Deren primärer Zweck ist also das Einüben und Trainieren bestimmter Fähigkeiten.

1.4. Kontext der Aufgabenstellung und Art des Feedbacks

Aufgaben können an Hochschulen in verschiedenen Kontexten gestellt werden, welche für die Art des Feedbacks entscheidend sind. Die folgenden Kontexte stellen eine grobe Übersicht dar:

- **Klausuren**: Em Ende einer Vorlesung werden meist schriftliche Klausuren gestellt, deren Korrekturergebnis über die Note und Bestehen der entsprechenden Veranstaltung entscheidet. Klausuren dienen in erster Linie der Leistungsbewertung, welche anhand der Antworten auf die in der in der Klausur gestellten Fragen vorgenommen wird. Es handelt sich hier um eine summative Bewertung, da die Leistungsbewertung am Ende des Lehr- und Lernprozesses erfolgt. Eine Variation der schriftlichen Klausur ist die mündliche Prüfung.
- **Übungsbetrieb**: Verbreitet ist das Stellen von Übungsaufgaben auch während der Vorlesungszeit. Dabei werden den Studierenden Aufgaben gestellt, die innerhalb einer bestimmten Frist bearbeitet werden müssen. Es erfolgt auch hier eine Korrektur durch Lehrpersonal. Diese Übungsaufgaben haben eine zweiseitige Kontrollfunktion. Einerseits können Studierende für sich selbst erkennen, wie gut sie den aktuellen Stoff beherrschen, andererseits erkennt auch der Dozent, ob es allgemeine Defizite gibt, die beispielsweise durch Lücken in der Vorlesung zustande kamen. Es ist auch üblich, nur solche Studierende zur Teilnahme an der Klausur zuzulassen, die die Aufgaben im Übungsbetrieb zu einem bestimmten Teil erfolgreich bearbeitet haben.
- **Freies Üben**: In diesem Kontext bearbeiten Studierende Aufgaben freiwillig, das heißt es besteht dazu keine formale Pflicht. Aufgaben stammen von Dozenten, aber

auch aus von Studierenden selbst erstellten Fragekatalogen, Aufgaben aus Lehrbüchern oder alten Klausuren. Studierende können hier entsprechend ihres selbst eingeschätzten Bedarfs üben, um so Lücken zu schließen. Es handelt sich dabei um eine Form des selbstregulierten Lernens. Feedback wäre auch hier wichtig, ist aber oft wegen mangelnden Lehrpersonals nur in Form von expliziten oder impliziten Musterlösungen in den Vorlesungsmaterialien verfügbar. Das gegenseitige Abfragen von Studierenden fällt ebenfalls in diesen Kontext, bei dem je nach Fähigkeiten des „Prüfers“ auch gutes Feedback gegeben werden kann. Natürlich wäre freies Üben prinzipiell auch mit Beteiligung von Lehrpersonal möglich, doch existieren entsprechende Angebote aufgrund der hohen Kosten in der Regel nicht.

In [Niegemann et al., 2008] wird Feedback definiert als „die von einem informationsverarbeitenden System als Folge eigener Verhaltensäußerungen wahrgenommenen Umgebungsveränderungen. Oder anders ausgedrückt: Das Ergebnis eines selbst initiierten Ereignisses dient als Information“. Informationsverarbeitende Systeme sind hier also Lernende, deren Aktionen die Umgebung verändern, wobei diese Veränderungen von Lernenden wahrgenommen werden. Ist dieses Feedback lernrelevant, wird es als informatives Feedback bezeichnet, welches folgende Formen hat [Narciss, 2006]:

- **Knowledge of performance** (KP): Nach der Bearbeitung der Aufgabe wird den Lernenden eine summative, meist quantitative Rückmeldung über die erreichte Leistung gegeben (z. B. Aufgabe zu x% korrekt gelöst)
- **Knowledge of result/response** (KR): Den Lernenden wird lediglich mitgeteilt, ob die eingegebene Lösung richtig oder falsch ist.
- **Knowledge of the correct answer** (KCR): Den Lernenden wird die korrekte Antwort angezeigt.
- **Answer until correct**(AUC) und **Multiple try feedback** (MTF): Nachdem eine Antwort falsch eingegeben wurde und entsprechendes Feedback erfolgte (KR) kann die Aufgabe erneut bearbeitet werden. Bei der Form AUC kann dieser Prozess fortgeführt werden, bis die korrekte Lösung eingegeben ist. Beim MTF ist die Anzahl der Wiederholungen hingegen beschränkt.
- **Elaborated feedback** (EF): Zusätzlich zu KR oder KCR werden den Lernenden Informationen gegeben, die helfen können, den oder die Fehler in Zukunft zu vermeiden. Darunter fallen auch Erklärungen, warum eine Lösung falsch oder korrekt ist.

In Klausuren wird in jedem Falle Feedback der Form KP gegeben, da dieses Ergebnis den formalen Erfolg der Teilnahme an der Veranstaltung abbildet. Für einzelne Aufgaben wird Feedback in der Form KR gegeben, auch werden den Lernenden meist die richtigen Lösungen zu den enthaltenen Fragen vorgelegt, dies aber oft nur in begrenztem zeitlichen Rahmen ohne die Möglichkeit, Aufgabenstellungen und Musterlösungen aufzubewahren und nachzubereiten. Noch flüchtiger ist das Feedback bei mündlichen Prüfungen. Hier lässt sich allerdings im Gegensatz zu schriftlichen Klausuren auch MTF einsetzen, wenn der Prüfer den Prüfling nach einer falschen Antwort noch weitere Antwortmöglichkeiten

einräumt. Feedback der Form EF wird hier aus pragmatischen Gründen auf individueller Basis meist nicht gegeben, da die Erstellung entsprechend aufwändig ist und nur mit zusätzlichem Personalaufwand geleistet werden kann.

Das im Übungsbetrieb gegebene Feedback ist ähnlich aufgebaut, wobei hier vermehrt Feedback der Form EF gegeben werden sollte, da nicht nur die Leistungsbewertung, sondern auch der Lernprozess wichtig ist. Dieses kann direkt Lernenden individuell durch schriftliche Kommentare bei der Rückgabe der Bearbeitungen oder in Kleingruppen in der Präsenzlehre gegeben werden. Feedback der Form AUC oder MTF ist hier unüblich, da die Aufgaben in der Regel in schriftlicher Form gestellt werden.

Für das freie Üben ist Feedback der Form KP zur besseren Einschätzung der eigenen Leistungsfähigkeit wichtig, wird in der Praxis aber selten gegeben. Wenn eine Musterlösung (Feedback der Form KCR) vorliegt, kann die eigene Lösung mit dieser verglichen werden. Inwieweit die Aufgabe damit gelöst ist, müssen Studierende also in der Regel selbst entscheiden. Feedback der Form AUC oder MTC ist hier kaum möglich, da die Aufgabe nach Ansicht der Musterlösung nicht mehr sinnvoll bearbeitet werden kann. Auch Feedback der Form EF müssen Lernende sich selbst ableiten.

Der Einsatz von E-Learning-Systemen hat für die Kontexte verschiedene Bedeutungen. Bei Klausuren ist der Einsatz noch unüblich, vor allem die Generierung des Feedbacks und damit die Entscheidung ob die Lösungen der Studierenden korrekt sind oder nicht, wird weitgehend menschlichen Korrektoren überlassen. Auch die jeweils geltende Prüfungsordnung kann ein Hindernis für den Einsatz von elektronischen Korrektursystemen darstellen. Etabliert haben sich hier teilweise Systeme zur automatischen Auswertung von Multiple-Choice-Klausuren.

Für Aufgaben im Übungsbetrieb hat E-Learning in der Form von Trainingssystemen einen potentiell großen Nutzen. Kann ein eingesetztes System Feedback in ähnlicher Weise wie eine Lehrperson generieren, so kann das Lehrpersonal sehr stark entlastet werden. Auch ist es hier bei geeigneter Konzeption möglich, Feedback der Formen AUC und MTF zu generieren.

Auch beim freien Üben übernimmt ein Trainingssystem die Aufgaben der Person, die Feedback generiert, was meist der Lernende selbst wäre. Lernende können also eine weit objektivere, und möglicherweise auch korrektere Rückmeldung zu ihren Lösungen erhalten.

1.5. Aufgabentypen

Die Komplexität von Konzeption und Implementierung solcher Trainingssysteme, und damit die Möglichkeit eines praktischen Einsatzes, hängt dabei stark von der Art der gestellten Aufgabe ab.

1.5.1. Klassifikation nach kognitiven Anforderungen

Aufgaben lassen sich zunächst dahingehend klassifizieren, welche kognitiven Operationen oder Anforderungen zur Bearbeitung nötig sind. Der dahingehend bekannteste Ansatz ist die Taxonomie der Lernziele von Benjamin Bloom [Bloom et al., 1956][Bloom, 1973]. Die Taxonomie gliedert kognitive Lernziele in sechs aufeinander aufbauende Bereiche. [Anderson und Krathwohl, 2001] verwenden in einer Überarbeitung der Bloom'schen Lernzieltaxonomie für die Bezeichnung der Lernziele durchgehend Verbformen und tauschen die letzten beiden Lernziele bezüglich ihrer Reihenfolge:

1. **erinnern** an Fakten, Methoden, Terminologien etc.
2. **verstehen** von Zusammenhängen Fakten, Methoden, Terminologien etc.
3. **anwenden** der Kenntnisse
4. **analysieren** von Sachverhalten und Problemstellungen
5. **evaluieren**, also Beurteilung von Sachverhalten
6. **erschaffen**, also die Verknüpfung von bekannter Information

Ein zentraler Aspekt dieser Taxonomie ist, dass Fähigkeiten eines Bereichs Fähigkeiten in den vorigen Bereichen voraussetzen. Auch [Körndle et al., 2004] nehmen eine Klassifizierung von kognitiven Anfordungen vor:

1. Erinnern
   a) *Recognition* - Abruf von Wissen mit Hinweisreiz (Wiedererkennen)
   b) *Recall* - Abruf von Wissen ohne Hinweisreiz (Reproduzieren)
2. Transformieren
   a) *Abbilden* - Darstellen von Inhalten in neuer Form
   b) *Paraphrasieren* - Wiedergeben von Inhalten mit anderen (eigenen) Worten
   c) *Illustrieren* - Finden von Beispielen
3. Klassifizieren
   a) *Diskriminieren* - Finden von Unterschieden
   b) *Generalisieren* - Finden von Gemeinsamkeiten
   c) *Kreuzklassifizieren* - Finden von Gemeinsamkeiten und Unterschieden
4. Argumentieren - Schlussfolgern
   a) *Extrapolieren* - Vorhersagen treffen, Hypothesen erstellen
   b) *Interpolieren* - Rückschlüsse auf einzelne Komponenten oder Faktoren ziehen, die einen Sachverhalt bestimmen
   c) *Interpretieren* - Deuten und Bewerten von Ergebnissen und Aussagen

Dabei bezeichnen [Körndle et al., 2004] diese Kategorien als allgemein gehalten und verweisen auf verschiedene themenspezifische Vorschläge, unter anderem auf den Ansatz von [Reif, 1995]. Dieser schlägt für die kognitiven Operationen bei naturwissenschaftliches Arbeiten die folgende Kategorisierung vor:

- Interpretation von Begriffen
- Beschreibung von Wissen
- Organisation des Wissens sowie
- Problemlösen

Für elektronische Trainingssysteme sind solche Einteilungen relevant, da deren Komponenten zur Feedback-Generierung für Aufgaben, die Operationen niedriger Ordnung verlangen, wesentlich einfacher zu konzipieren und implementieren sind. Dies gilt vor allem für die Kategorien „Kenntnis“ nach Bloom, „Erinnern“ nach Körndle et al. und für die „Interpretation von Begriffen“ nach Reif. Lösungen für entsprechende Aufgaben sind relativ leicht maschinell zu bewerten. Dabei handelt es sich vorwiegend um geschlossene oder halb-offene Aufgaben, deren Form im nächsten Abschnitt beschrieben wird.

1.5.2. Klassifikation nach Form

Aufgaben lassen sich auch anhand ihrer Form in verschiedene Klassen oder Typen einteilen: [Rütter, 1982] unterteilt in die drei Klassen der „offenen, halboffenen und geschlossenen Testaufgabe“, während [Lienert und Raatz, 1998] in zwei Typen, die freie und die gebundene Aufgabenbeantwortung, gliedern. Beide Ansätze basieren auf der gleichen Idee, die Aufgabentypen danach zu unterscheiden, wie viel Freiheit die Lernenden bei der Wahl ihrer Antwort haben. Im Rahmen dieser Arbeit wird die Klassifizierung nach Rütter verwendet. Diese basiert darauf, ob die möglichen Lösungen einer Aufgabe dem „Auswerter“ und/oder den Lernenden vorgegeben sind. Der „Auswerter“ ist dabei die Person, welche die Antwort auf die Frage auswertet bzw. bewertet und die Rückmeldung generiert. Im Kontext von elektronischen Trainingssystemen kann der Auswerter auch eine Maschine bzw. eine Software sein. Die Grobstruktur[1] der Klassifikation nach Rütter ist wie folgt:

- Bei **geschlossenen Testaufgaben** ist die Antwort sowohl den Lernenden als auch dem Auswerter vorgegeben. Der Lernende selektiert hier also seine Antwort(en) aus einer vorgegebenen Menge von Antworten. Darunter fallen beispielsweise klassische Multiple-Choice-Aufgaben, aber auch Zuordnungsaufgaben, bei denen Lernende Elemente zweier (meist gleichmächtiger) Mengen einander zuordnen müssen.
- Bei **halb-offenen Aufgaben** ist die Antwort nicht den Lernenden, wohl aber dem Auswerter vorgegeben. Lernende müssen hier meist selbst eine Antwort erstellen, beispielsweise bei sogenannten „Short-Answer“-Formaten, bei denen ein sehr kurzer Text, ein Wort oder eine Zahl eingegeben werden muss. Lückentexte gehören ebenso in diese Kategorie. Eine auch in diese Kategorie fallende und nur in E-Learning-Systemen sinnvoll einsetzbare Erweiterung von Multiple-Choice-Fragen stellen sogenannte „Long-Menu“-Fragen dar. Hier können Lernende zwar aus einer

[1] Rütter unterteilt diese Kategorien noch weiter. Diese Feinstruktur ist im Rahmen dieser Arbeit jedoch nicht relevant.

(optional hierarchisch angeordneten) Menge vorgegebener Antworten wählen, doch ist diese Menge so groß, dass es praktisch nicht möglich ist, die Menge möglicherweise richtiger Antworten durch das Ausschließen offensichtlich falscher Antworten einzugrenzen. Die unkomplizierte Auswahl einer Antwort kann bei hierarchisch angeordneten Mengen durch entsprechende Navigation oder durch eine Suchfunktion ermöglicht werden.

- Bei **offenen Aufgaben** ist die Antwort weder den Lernenden noch dem Auswerter vorgegeben. Das bekannteste Beispiel sind hier Essay-Aufgaben, bei denen Lernende zur Lösung der Aufgabe einen langen Text eingeben müssen.

Bei den offenen Aufgaben ist die fehlende Vorgabe einer Lösung für den Auswerter so zu verstehen, dass dieser nicht alle möglichen korrekten Lösungen für diese Aufgabe kennt. Der Auswerter bzw. der Autor der Aufgabe ist aber sehr wohl in der Lage, eine oder mehrere korrekte Lösungen für die Aufgabe zu erstellen. Die Existenz von Musterlösungen zu offenen Aufgaben ist also kein Widerspruch.

Geschlossene Aufgaben können von elektronischen Trainingssystemen sehr gut verarbeitet werden. Dem System sind alle möglichen Antworten bekannt, ebenso deren Korrektheitsgrad, also inwieweit die Antwort korrekt ist. Oftmals ist dies eine binäre Abbildung, bei der die Antworten entweder völlig falsch oder völlig richtig sind. Diese Fragen eignen sich sehr gut, um Faktenwissen abzubilden, also solche Lernziele zu behandeln, die kognitive Operationen niedriger Ordnung verlangen. Zwar lassen sich mit geschlossenen Aufgaben, beispielsweise mit Multiple-Choice-Fragen auch kognitive Operationen höherer Ordnungen abfragen [Haladyna, 2004], doch beschränkt sich die letztlich ausschlaggebende Aktion des Lernenden stets nur auf die Auswahl aus einer Menge vorgegebener Alternativen, was nur sehr eingeschränkt als praxisnah anzusehen ist.

Bei halb-offenen Aufgaben kommt es stark auf den konkreten Aufgabentyp an, ob sie leicht maschinell auswertbar sind oder nicht. Long-Menu-Fragen lassen sich genauso leicht behandeln wie geschlossene Aufgaben, da die Anzahl der Antworten für eine Maschine keine Rolle spielt. Short-Answer-Fragen können im Falle einer geforderten Eingabe einer Zahl ebenfalls sehr gut ausgewertet werden. Hier können, so vom Autor der Frage gewünscht, auch nicht völlig korrekte Antworten noch mit teilweise positivem Feedback versehen werden. Wird beispielsweise nach einer Jahreszahl gefragt und ist für den Autor der Frage das genaue Datum zweitrangig, kann hier mit einer zuvor festgelegten Toleranz auch dann positives Feedback gegeben werden, wenn Lernende eine abweichende Jahreszahl angeben. Bei der Eingabe von ein oder mehreren Wörtern ist es wichtig, dass Trainingssysteme eine gewisse Toleranz zulassen, da Antworten andernfalls bei geringer Variation oder Tippfehlern als falsch gewertet werden, was auf die Lernenden demotivierend wirken kann.

Offene Aufgaben lassen sich im Allgemeinen von Trainingssystemen nur sehr schwer automatisch auswerten. Ursache dafür ist bei der Eingabe freier Texte die Verwendung von natürlicher Sprache, die wegen ihrer Komplexität nur schwer maschinell interpretierbar ist. Große Probleme sind hier unter anderem Mehrdeutigkeiten und die implizite Verwendung von Hintergrundwissen, welches bei Menschen vorausgesetzt werden kann,

Maschinen jedoch nicht verfügbar ist. Eine weitere, in diesem Kontext negative Eigenschaft der natürlichen Sprache ist die Möglichkeit der Paraphrasierung. So gibt es sehr viele Möglichkeiten, einen Sachverhalt auf verschiedene Weise auszudrücken. Menschen können dies gut erkennen, Maschinen hingegen nicht. [Eilers, 2006] untersuchte 16 Lernplattformen unter anderem auf ihre Möglichkeiten im Umgang mit Freitextfragen: In 12 dieser Systeme ließen sich Freitextfragen stellen. Jedoch konnte keines die entsprechenden Antworten vollständig selbst auswerten. Da sich mit diesem Aufgabentyp auch sehr praxisnahe Aufgaben stellen lassen und eine Bewertung durch Lehrpersonal deutlich zeitaufwändiger ist als bei geschlossenen und halb-offenen Aufgaben, ist die vermeintlich beschränkte Leistungsfähigkeit von Trainingssystemen in diesem Kontext besonders bedauerlich. Als Lösungsmöglichkeit wird in [Niegemann et al., 2008] vorgeschlagen, detaillierte Musterlösungen einzusetzen, also zumindest Feedback der Form KCR zu geben, so dass die Lernenden ihre Lösung selbst mit der Musterlösung vergleichen und sich so eine Rückmeldung bezüglich der Korrektheit ihrer Lösung ableiten können.

Im Folgenden Abschnitt wird eine weitere Dimension der Klassifizierung von Aufgaben erläutert. Die daraus abgeleiteten Eigenschaften von Aufgaben sind können Trainingssysteme helfen, entsprechende Aufgaben doch gut auswerten zu können.

1.5.3. Klassifikation nach Strukturierung

Information, Wissen oder Daten im Allgemeinen, können in strukturierter oder unstrukturierter Form vorliegen. Als unstrukturierte Daten, für die es eben keine streng formalisierte Struktur gibt, gelten beispielsweise Dokumente in natürlicher Sprache, aber auch Tonaufnahmen oder Bilder. Sie sind maschinell gegenwärtig nicht mit trivialen Mitteln durchsuchbar, was bedeutet, dass es nicht möglich ist, gezielt Informationen aus diesen Daten zu gewinnen. Strukturierte Daten dagegen liegen in einer bestimmten Anordnung vor deren Struktur formal beschrieben werden kann. Dazu gehören relationale Datenbanken oder Dokumente in formalen Sprachen, also beispielsweise Programmiersprachen. Auch Diagramme, sofern sie einer wohldefinierten Syntax folgen, gehören zu strukturierten Daten. Diese Eigenschaft ist jedoch nicht binär. Es existieren ebenso Daten, die nur halb strukturiert sind. Diese werden als semi-strukturiert bezeichnet. Auch wird von schwach, stark oder sehr stark strukturierten Daten gesprochen.

Aufgaben können nun danach klassifiziert werden, in welcher Form eine zugehörige Lösung eingegeben wird. Lösungen zu geschlossenen Aufgaben sind stets strukturiert, da quasi ein aufgabenspezifischer, impliziter Formalismus besteht, der vorgibt, wie die Lösung einer Aufgabe abgebildet werden kann. Bei Multiple-Choice-Fragen kann eine Lösung beispielsweise durch eine einzige Zahl abgebildet werden, sofern den verfügbaren Lösungen zuvor jeweils eineindeutig eine Zahl zugeordnet wurde. Bei halb-offenen Aufgaben kommt es auch hier wieder auf den konkreten Aufgabentyp an. Die Antwort auf Long-Menu-Fragen ist ebenso stark strukturiert wie die Antwort auf eine Short-Answer-Frage, bei der nur die Eingabe einer Zahl erlaubt ist. Wird die Eingabe von mehreren Wörtern erwartet, ist die Antwort im Allgemeinen nicht strukturiert.

Lösungen für die oben genannten Beispiele für offene Aufgaben liegen in unstrukturierter Form vor. Es existieren allerdings auch Aufgabentypen, bei denen von den Lernenden stärker strukturierte Lösungen konstruiert werden müssen.
Eine den Freitextfragen verwandte Form sind sogenannte Lösungsskizzen. Dabei wird zu einer Aufgabe, die normalerweise mit einem längeren Text zu beantworten ist, nur eine skizzierte Lösung in hierarchischer Struktur verlangt. Die Knoten in dieser Baumstruktur enthalten wenige Wörter bis wenige Sätze. Diese Lösungsskizzen dienen der Vorbereitung längerer Texte und gelten als intellektuelle Hauptleistung. Sie sind verbreitet bei der Ausbildung von Juristen, finden aber auch in anderen Fachgebieten Verwendung, beispielsweise in der Didaktik, wenn den Lernenden die Aufgabe gestellt wird, eine Unterrichtsstunde zu planen. Grundsätzlich lässt sich jede Essay-Aufgabe, in der eine hierarchische Struktur von Konzepten konstruiert wird als Lösungsskizzen-Aufgabe stellen.
Ein weiterer Aufgabentyp sind sogenannte Assoziogramme in der sprachlichen Ausbildung. Dabei steht ein Begriff im Mittelpunkt, zu dem die Lernenden weitere Begriffe finden müssen, die in natürlicher Sprache lediglich „oft mit dem gegebenen Begriff gemeinsam auftreten“. Wichtig ist hier, dass bei einer Antwort ausgezeichnet ist, welche Begriffe in Beziehung zueinander stehen und welcher Art diese Beziehung ist. Zu stark strukturierten Aufgabentypen zählen auch solche, die mathematische Formeln oder chemische Verbindungen als Antwort haben. Auch Programmcode, UML-Diagramme und Datenbank-Anfragesprachen sind stark strukturierte Lösungen.

Strukturierte Aufgaben lassen sich für den Gebrauch in elektronischen Trainingssystemen außerdem noch dadurch unterscheiden, ob Lernende Lösungen eingeben können, die gegenüber dem zugrunde liegenden Formalismus nicht korrekt sind. So kann beispielsweise beim Zeichnen von Diagrammen das Erstellen syntaktisch unkorrekter Strukturen technisch verhindert werden. Da diese Einschränkung für Lernende allerdings auch schon eine Form des Feedbacks darstellt, das implizit gegeben wird, ist diese Unterscheidung praktisch nicht relevant.

Für elektronische Trainingssysteme bieten in strukturierter Form vorliegende Lösungen einen immensen Vorteil, da daraus gezielt Informationen gewonnen werden können. Entsprechende Aufgaben werden im Folgenden zur Vereinfachung „strukturierte Aufgaben“ genannt.

1.5.4. Klassifikation nach maschineller Interpretierbarkeit

Eine Unterklasse von strukturierten Aufgaben bilden solche Aufgaben, deren Lösungen in irgendeiner Weise maschinell interpretierbar sind, wenn sie den entsprechenden formalen Vorgaben genügen. Unter maschineller Interpretierbarkeit wird hier die Eigenschaft einer Lösung verstanden, insofern von einer Maschine interpretiert werden zu können, als dass die Interpretation ein gewisses Ergebnis liefert, welches durch die Lösung nur implizit angegeben wird. Dabei ist das Ergebnis ebenfalls strukturiert. Darunter zählen beispielsweise Aufgaben, deren Antwort die Form von Programmcode hat. Auch bestimmte Diagramme, zum Beispiel UML-Klassen- und Aktivitätsdiagramme, sind nach

Übersetzung in Programmcode maschinell interpretierbar. Ebenso verhält es sich mit Datenbank-Anfragesprachen wie SQL. Auch mathematische Formeln sind je nach Aufgabenstellung maschinell interpretierbar, zum Beispiel wenn mit der eingegeben Formel direkt etwas berechnet werden kann. Diesen Aufgabentypen ist gemein, dass die Basis der technischen Lösungen für die Interpretation der Antworten in Form von Compilern, Interpretern, Datenbank Management Systemen (DBMS) oder Mathematik-Programmen meist bereits existiert, was die Implementierung entsprechender Trainingssysteme erleichtert. Aufgaben mit maschinell interpretierbaren Lösungen werden im Folgenden zur Vereinfachung „maschinell interpretierbare Aufgaben“ genannt.

In den folgenden Abschnitten wird erläutert, wie die Eigenschaften von strukturierten und maschinell interpretierbaren Aufgaben bei der automatischen Feedback-Generierung ausgenutzt werden können.

1.6. Ansätze zur automatischen Feedback-Generierung für strukturierte Aufgaben

Durch eine Strukturierung der Antworten sind diese maschinell leichter zu verarbeiten. Die Antwort ist durch die Struktur segmentierbar, die Segmente sind ggf. ausgezeichnet und stehen in einer bestimmten Beziehung zueinander. Diese Teilantworten können maschinell besser verarbeitet und ausgewertet werden als eine unstrukturierte Antwort. Das generierte Feedback kann sich dann auf diese Teilantworten beziehen, was die Verständlichkeit für die Lernenden erhöht, da Fehler besser lokalisiert werden. Es handelt sich also um ein Verfahren im Sinne des „Teile und Herrsche“-Ansatzes. Besonders hilfreich ist hier die Verwendung einer oder mehrerer Musterlösungen als Vergleich. Diese Strategie bildet letztlich ein auch von menschlichen Korrektoren verwendetes Verfahren ab, in welchem eine Musterlösung in eine Menge von Teillösungen aufgespalten wird und die zu korrigierenden Lernerlösungen dahingehend untersucht werden, ob diese Teillösungen dort (in ähnlicher Weise) ebenfalls vorkommen.

Die grundlegende Strategie des Verfahrens besteht also darin, eine Lernerlösung mit Musterlösungen zu vergleichen und jeweils ein „Überdeckungsmaß“ (zwischen 0 und 100 %) zu bestimmen. Die Musterlösung, welche zur Lernerlösung das höchste Überdeckungsmaß aufweist, gilt als Referenz und wird den Lernenden als Musterlösung im Feedback angezeigt (Feedback der Form KCR). Der Zahlenwert der Überdeckung zu dieser Referenzlösung entspricht ebenso der Bewertung der Lernerlösung (Feedback der Form KP). Außerdem wird den Lernenden als weiteres Feedback ein Dokument angezeigt, in welchem erklärt wird, wie das Überdeckungsmaß, und somit die Bewertung der eigenen Lösung, zustande gekommen ist (Feedback der Form EF). Muster- und Lernerlösung können nun als Menge von Teillösungen interpretiert werden, so dass ein Überdeckungsmaß und ein daraus abgeleitetes Ähnlichkeitsmaß allgemein definiert werden kann.

1.6.1. Überdeckungsmaß von Mengen

Das im Rahmen dieser Arbeit entwickelte Überdeckungsmaß U ist eine Funktion über zwei Mengen mit einem Wertebereich zwischen 0 und 1, das sich am Konzept der Teilmenge aus der Mengenlehre orientiert: Eine Menge A ist genau dann eine Teilmenge der Menge B, wenn für alle Elemente $a \in A$ auch $a \in B$ gilt. Dieser Ansatz wird nun insofern erweitert, als dass nicht nur überprüft wird, ob ein Element in identischer Weise in der anderen Menge vorkommt, sondern ob ein „ähnliches“ Element vorkommt, wobei ein bestimmtes Ähnlichkeitsmaß verwendet wird. Die Ähnlichkeit S zwischen zwei Elementen a und b ist eine Funktion mit einem Wertebereich zwischen 0 und 1. Als Einschränkung gilt, dass die Elemente beider Mengen gemeinsamen „Typs“ sind, sie also jeweils Element einer Menge E sind, die alle Elemente des Typs enthält (und nur diese). Sei M die Menge aller Mengen, die ausschließlich Elemente aus E enthalten. Es gilt:

$$\mathbb{R}^{01} = \{r \mid r \in \mathbb{R}, 0 \leq r \leq 1\} \tag{1.1}$$

$$\forall m \in M : x \in m \implies x \in E \tag{1.2}$$

$$U : M \times M \to \mathbb{R}^{01} \tag{1.3}$$

$$S : E \times E \to \mathbb{R}^{01} \tag{1.4}$$

Seien $A, B \in M$, beide Mengen beinhalten also ausschließlich Elemente gleichen Typs. Sei $a \in A$ und $b \in B$. Zur Berechnung des Überdeckungsmaßes wird nun eine eineindeutige Zuordnung zwischen Elementen aus A und B mit der Eigenschaft gesucht, dass die Summe der Ähnlichkeiten der einander zugeordneten Elemente maximiert wird. Es handelt sich also um ein lineares Zuordnungsproblem, dessen Kosten- oder Gewichtsfunktion[2] das entsprechende Ähnlichkeitsmaß ist. Die Lösung des Zuordnungsproblems ist eine Bijektion Z mit der Eigenschaft, dass die Nutzenfunktion K maximiert ist.

$$Z : A \to B \tag{1.5}$$

$$K(A, B, Z) = \sum_{a \in A} S(a, Z(a)) \tag{1.6}$$

Die Gewichtsfunktion des linearen Zuordnungsproblems, die jedem Paar (a, b) ein Gewicht zuordnet, ist also S. Da sich ein lineares Zuordnungsproblem nur bei Mengen gleicher Mächtigkeit lösen lässt, wird die Menge mit weniger Elementen zuvor mit „Dummy-Elementen“ aufgefüllt, deren Ähnlichkeit zu jedem $e \in E$ gleich 0 ist. Das Zuordnungsproblem lässt sich dann beispielsweise mit der Ungarischen Methode [Kuhn, 1955] in polynomialer Laufzeit optimal lösen. Ist das Zuordnungsproblem gelöst, ist jedem $a \in A$

[2] Oft wird bei linearen Zuordnungsproblemen von einer Kostenfunktion gesprochen, die minimiert wird. Der Ansatz ist analog, doch passt hier eine zu maximierende Nutzenfunktion besser, da eine maximale kumulierte Ähnlichkeit angestrebt wird.

ein Element $b \in B$ zugeordnet, und jedes dieser Paare besitzt eine Ähnlichkeit zwischen 0 und 1. Zur Berechnung des Überdeckungsmaßes werden diese Ähnlichkeiten (identisch zur Nutzenfunktion) aufaddiert und durch Division mit der Mächtigkeit von A normalisiert. Das Überdeckungsmaß $U(A,B)$ für $A \neq \varnothing$ ist also:

$$U(A, B) = \frac{\sum_{a \in A} S(a, Z(a))}{|A|} \tag{1.7}$$

Falls $A = \varnothing$ und $B \neq \varnothing$, so ist $U(A, B) = 0$. Falls $A = B = \varnothing$, so ist $U(A, B) = 1$. Das Überdeckungsmaß ist also für identische Mengen stets 1, für völlig verschiedene Mengen, also Mengen deren paarweise Elemente eine Ähnlichkeit gleich 0 haben, ist das Überdeckungsmaß stets 0. Außerdem ist das Überdeckungsmaß bei Mengen verschiedener Mächtigkeit nicht kommutativ. Da $S(a, Z(a))$ stets zwischen 0 und 1 liegt, liegt auch $U(A, B)$ zwischen 0 und 1.

Beispiel: Sei $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2\}$ und $a_1 = b_1$. B wird um ein Dummy-Element b_d erweitert. Die Ähnlichkeiten $S(a_i, b_j)$ seien wie folgt:

	a_1	a_2	a_3
b_1	1,0	0,9	0,1
b_2	0,9	0,7	0
b_d	0	0	0

Das Zuordnungsproblem wird gelöst und die optimale Zuordnung mit folgenden Paaren wird gefunden: $\{(a_1, b_2), (a_2, b_1), (a_3, b_d)\}$, da hier die Kosten mit einem Wert von 1,8 maximal sind. Das Überdeckungsmaß $U(A,B)$ ist nun:foo

$$U(A,B) = \frac{S(a_1,b_2)+S(a_2,b_1)+S(a_3,b_d)}{3} = \frac{0{,}9+0{,}9+0}{3} = 0{,}6$$

Die Menge A wird also in einem Maß von 0,6 von der Menge B überdeckt. Werden die Argumente vertauscht, wird also $U(B, A)$ berechnet, erhält man dagegen ein Überdeckungsmaß von 0,9, da hier zwar die gleiche Zuordnung stattfindet, aber durch Division mit der Mächtigkiet von B normiert wird:

$$U(B,A) = \frac{S(a_1,b_2)+S(a_2,b_1)+S(a_3,b_d)}{2} = \frac{0{,}9+0{,}9+0}{2} = 0{,}9$$

Diese Ergebnisse erscheinen intuitiv sinnvoll: A wird von B zu knapp zwei Dritteln überdeckt, da nur 2 der 3 in A enthaltenen Elemente (fast) identisch in B vorkommen. B wird von A dagegen fast völlig überdeckt, da beide in B enthaltenen Elemente (fast) identisch in A vorkommen.

Die Elemente, für die ein Überdeckungsmaß berechnet werden soll, können selbst wiederum Mengen sein. Daher muss ein Ähnlichkeitsmaß für Mengen definiert werden.

1.6.1.1. Ähnlichkeit von Mengen

In der Mengenlehre sind zwei Mengen A und B identisch, wenn eine Menge jeweils Teilmenge der anderen ist, wenn also $A \subseteq B$ und $B \subseteq A$ gilt. Zur Berechnung der Ähnlichkeit der Mengen $S(A, B)$ werden nun analog zum Ansatz der Berechnung des Überdeckungsmaßes, die aus der Teilmengeneigenschaft abgeleitet ist (siehe Abschnitt 1.6.1), die Überdeckungsmaße $U(A, B)$ und $U(B, A)$ berechnet. Zur Normalisierung wird das arithmetische Mittel gebildet.

$$S(A, B) = \frac{U(A, B) + U(B, A)}{2} \tag{1.8}$$

Das Ähnlichkeitsmaß von Mengen ist im Gegensatz zum Überdeckungsmaß also kommutativ. Zwar existieren schon Ähnlichkeitsmaße von Mengen in der Literatur, doch berücksichtigen diese nur die Gleichheit der Elemente, nicht deren Ähnlichkeit. Der **Jaccard-Koeffizient** zweier Mengen A und B ist definiert als:

$$S_J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1.9}$$

Es wird hier die Anzahl der gemeinsamen, also identischen Elemente gezählt, anschließend wird durch Division mit der Mächtigkeit der Vereinigungsmenge normiert. Für oben genanntes Beispiel ist das vorgestellte Ähnlichkeitsmaß 0,75, der Jaccard-Koeffizient hingegen nur 0,25, da eben nur die Gleichheit von a_1 und b_1, nicht aber die Ähnlichkeit der anderen Elemente berücksichtigt wird. Ein anderes Maß ist der **Sørensen-Index**, der definiert ist als:

$$S_S = \frac{2 \cdot |A \cap B|}{|A| + |B|} \tag{1.10}$$

Auch dieser berücksichtigt Ähnlichkeiten von Elementen nicht und liefert für genanntes Beispiel den Wert von 0,4.

1.6.2. Regelbasierte Überprüfung

Wie beschrieben können strukturierte Lösungen in Teile aufgespalten und separat untersucht werden. Für strukturierte, aber nicht maschinell interpretierbare Aufgaben kann eine regelbasierte Überprüfung bei der Feststellung helfen, ob die Lernerlösung den formalen Ansprüchen genügt oder nicht, so dass bei formaler Unkorrektheit Feedback der Formen KR und EF gegeben werden kann. Auch bei formaler Korrektheit kann positives Feedback der Form KR gegeben werden, wobei sich dieses eben nur auf korrekte Syntax, nicht auf die Korrektheit der Lösung insgesamt bezieht.

Neben dieser Überprüfung auf syntaktischer Ebene kann der regelbasierter Ansatz auch auf semantischer Ebene verwendet werden, indem Lernerlösungen auf das Vorkommen oder Nicht-Vorkommen bestimmter Elemente mit bestimmten Eigenschaften hin untersucht werden. Zusammen mit einem möglichen Feedback der Form KP kann den Lernenden dadurch auch Feedback des Typs EF gegeben werden. Dazu werden die entsprechend verletzten Regeln in irgendeiner Form verbalisiert, möglicherweise auch in natürlicher Sprache. Wird durch die Regeln implizit eine Musterlösung definiert, kann auch Feedback der Form KCR gegeben werden.

Ein Beispiel, bei dem eine regelbasierter Überprüfung sehr gut verwendet werden kann, ist die Untersuchung von Programmcode auf sogenannten „Code Smells“ oder „Bad Smells“ [Fowler und Beck, 2000], bei welchen es sich um negative Eigenschaften des Programmcodes handelt, die gewissen Konventionen widersprechen und einen Umbau nahe legen (siehe auch Abschnitt 3.2).

Auch für Diagramme ist eine solche regelbasierte Überprüfung mit entsprechendem Feedback geeignet. Dabei können Eigenschaften, wie die Anzahl an Elementen oder die räumliche Ausdehnung abgefragt werden. Auch der Stil kann überprüft werden, beispielsweise ob die Benamung bestimmter Elemente einer Konvention folgt, beispielsweise der Verwendung von Großschreibung. Bei Aufgaben, deren Lösungen natürlichsprachigen Text enthalten, also beispielsweise bei Lösungsskizzen, können hier Prüfungen auf Rechtschreibung und Grammatik stattfinden und zur Generierung des Feedbacks herangezogen werden.

1.7. Ansätze zur automatischen Feedback-Generierung für maschinell interpretierbaren Aufgaben

Wie zuvor beschrieben, können Lösungen von diesen Aufgaben maschinell interpretiert, also ausgeführt werden. Als Grundlage zur Feedback-Generierung können nun zwei Aspekte dieses Vorgangs herangezogen werden.

1.7.1. Ergebnisvergleich

Die maschinelle Interpretierbarkeit kann auf unterschiedliche Weise zu einem Vergleich der Ergebnisse der Interpretation bzw. Ausführung genutzt werden. Was genau verglichen wird, hängt zunächst davon ab, ob zur Ausführung einer Lösung Eingabedaten benötigt werden. Entsprechende Aufgaben werden im Folgenden „datenverarbeitende Aufgaben“ genannt.

Bei datenverarbeitenden Aufgaben müssen zur Generierung von Feedback Eingabedaten und entweder zu diesen Eingabedaten entsprechend korrekte Ergebnisse oder Musterlösungen vorliegen. Sei L die Menge aller Lösungen, die den formalen Anforderungen zur Ausführung genügen. Sei D die Menge möglicher Eingabedaten und die Ausführung $A_{LD} : L \times D \mapsto E$ eine Funktion über L und D, die zwei entsprechenden Elementen ein

Ergebnis $e \in E$ zuordnet, also aus einer Lösung und Eingabedaten ein Ergebnis berechnet. Die Feedback-Generierung $G : E \times E \mapsto F$ ist dann eine Funktion, die einem Tupel von Ergebnissen ein Feedback $f \in F$ zuordnet, wobei das erste Argumente dabei stets ein korrektes Ergebnis, das zweite das Ergebnis einer Lernerlösung ist.

Liegen bei datenverarbeitenden Aufgaben also keine Musterlösungen vor, so müssen als Hintergrundwissen zu den Eingabedaten entsprechend korrekte Ausgabedaten, also Ergebnisse aus der Menge E angegeben werden. Es existiert formal also eine Funktion $H : D \mapsto E$, die jedem Eingabedatum ein korrektes Ergebnis zuordnet. Sei $l_l \in L$ eine Lernerlösung und seien $d \in D$ Eingabedaten, so kann das Feedback hier wie folgt generiert werden:

$$f_1 = G(H(d), A_{LD}(l_l, d)) \tag{1.11}$$

Liegt dagegen eine Musterlösungen vor, kann sie die explizite Angabe von korrekten Ergebnissen zu Eingabedaten ersetzen. Statt dessen wird die Musterlösung mit vorliegenden Eingabedaten ausgeführt, wobei nun dieses Ergebnis mit dem Ergebnis der Ausführung der Lernerlösung verglichen wird, so dass auch ohne die explizite Angabe von richtigen Ergebnissen Feedback generiert werden kann:

$$f_2 = G(A_{LD}(l_m, d), A_{LD}(l_l, d)) \tag{1.12}$$

Hier ist ein Vorteil, dass bei einer Änderung an Eingabedaten keine neuen Ausgabedaten berechnet und statisch hinterlegt werden müssen, da diese während der Feedback-Generierung stets dynamisch generiert werden.

Handelt es sich nicht um datenverarbeitende Aufgaben, kann ebenfalls ein Ergebnisvergleich stattfinden. Die Funktion zur Ausführung einer Lösung hat hier nur ein Argument, ist also der Form $A_L : L \mapsto E$. Um einen Ergebnisvergleich zu ermöglichen, muss also mindestens ein korrektes Ergebnis $e \in E$ oder eine Musterlösung angegeben werden. Feedback wird nun generiert durch:

$$f_3 = G(e, A_L(l_l)) \tag{1.13}$$

oder

$$f_4 = G(A_L(l_m), A_L(l_l)) \tag{1.14}$$

Der letztliche Ergebnisvergleich, also die Generierung eines Feedbacks, kann bei all diesen Varianten durch die Lernenden selbst, aber auch maschinell erfolgen. Dies hängt von dem konkreten Aufgabentyp und beispielsweise auch von der Anzahl den möglicherweise hinterlegten Eingabedaten und Ergebnissen ab: Eine sehr große Menge von Daten bzw. Ergebnissen kann von Menschen nur sehr schwer, von Maschinen (bis zu einem gewissen Grad) sehr einfach verglichen werden.

Zur Veranschaulichung folgt für jeden Variante ein Beispiel:

- **Datenverarbeitende Aufgaben ohne Musterlösung**: Ein klassisches Beispiel für diese Art des Ergebnisvergleichs sind sogenannte Unit-Tests in Trainingssystemen zur Programmierausbildung. Dabei wird in der Aufgabenstellung verlangt, ein Programm zu entwickeln oder es so zu ergänzen, dass es bei bestimmten Eingaben bestimmte Ausgaben liefert. In den Unit-Tests sind die korrekten Ergebnisse explizit statisch oder implizit hinterlegt. In letzterem Fall benutzt der Unit-Test eine Routine, die korrekte Ergebnisse liefert, die aber nicht als Musterlösung geeignet ist, beispielsweise da bestimmte Funktionen der Programmiersprache von den Lernenden nicht verwendet werden sollen. Die Lernerlösung wird dann mit mehreren hinterlegten Eingabedaten ausgeführt und es wird überprüft, ob die Ausführung die erwarteten Ergebnisse liefert.
- **Datenverarbeitende Aufgaben mit Musterlösung**: Ein entsprechendes Beispiel sind Aufgaben für Datenbankanfragesprachen, wobei eine Anfrage einer Lösung entspricht und die Eingaben in Form einer gefüllten Datenbank vorliegen. Musterlösung und Lernerlösung werden ausgeführt und die Ergebnistabelle bzw. der veränderte Zustand der Datenbank dient als zu vergleichendes Ergebnis.
- **Nicht datenverarbeitende Aufgaben**: Eine möglicher Aufgabentyp ist hier die Eingabe von statischen HTML-Seiten. Das gewünschte Ergebnis könnte eine Webseite bestimmter Optik sein, die beispielsweise als Screenshot oder als Musterlösung in Form eines HTML-Dokuments vorliegt. Das von einer sogenannten Rendering-Engine[3] visualisierte Dokument der Lernerlösung kann mit dem hinterlegten Screenshot oder dem visualisierten Musterlösungs-Dokument verglichen werden.

1.7.2. Visualisierung der Ausführung

Ist die Aufgabe und das entsprechende Trainingssystem geeignet gestaltet, kann die Ausführung der Lernerlösung visualisiert werden, also in irgendeiner Form bildhaft dargestellt werden. Bei geeigneter Visualisierung können Lernende so beispielsweise erkennen, an welcher Stelle ihre Lösung zu einem ungewünschten Verhalten führt. Dies entspricht Feedback der Form EF. Ein Beispiel ist hier die Ausführung von Programmcode, sofern damit visualisierbare Vorgänge abgebildet werden. Naheliegend ist hier die Visualisierung in Form einer generierten Animation.

1.8. Trainingssysteme für offene, strukturierte Aufgaben

Die hier vorgestellten Konzepte wurden eingesetzt, um vier Aufgabentypen und zugehörige webbasierte Trainingssysteme bzw. entsprechende Komponenten zu konzipieren und zu implementieren, bei welchen die Eigenschaften der starken Strukturierung und

[3] Rendering-Engines sind Softwarekomponenten, die vor allem in Web-Browsern eingesetzt werden. Sie können aber auch in Trainingssysteme integriert werden.

der maschinellen Interpretierbarkeit ausgenutzt werden. Entsprechende Aufgaben haben eine besonders große Verbreitung in der Informatik und verwandten Fachgebieten, da hier in besonderem Maße mit strukturierten Daten umgegangen wird. Drei der vier im Folgenden vorgestellten Systeme sind der Informatik zuzuordnen, während ein Trainingssystem in verschiedenen Fachgebieten, vor allem in der Medizin, eingesetzt werden kann.

Mit der in Kapitel 2 beschriebenen Komponente des Trainingssystem CaseTrain [Hörnlein et al., 2009] kann die Erstellung von **UML Klassendiagrammen** eingeübt werden. Es verwendet zur Feedback-Generierung ein Überdeckungsmaß von Klassendiagrammen, welches aus den hier vorgestellten Überdeckungs- und Ähnlichkeitsmaßen von Mengen abgeleitet ist.

In Kapitel 3 wird ein Trainingssystem „WARP“ zur Erstellung von **UML Aktivitätsdiagrammen** vorgestellt, das deren Eigenschaft der Übersetzbarkeit in Programmcode und damit deren maschinelle Interpretierbarkeit bei der Generierung von Feedback ausnutzt. Die Ausführung des erzeugten Programmcodes wird visualisiert und dient den Lernenden zusammen mit einem automatischen Ergebnisvergleich als Feedback.

SQL Anfragen können mit dem in Kapitel 4 beschriebenen Trainingssystem „ÜPS“ eingeübt werden. Dabei findet nach einer durch das DBMS vorgenommenen syntaktischen Prüfung und entsprechender Rückmeldung ein statischer Vergleich mit hinterlegten Musterlösungen statt, der die Lernenden auf fehlende oder fehlerhafte Elemente ihrer Lösung hinweist. Musterlösung und Lernerlösung werden anschließend auf einer gefüllten Datenbank ausgeführt, der Vergleich der entsprechenden Ergebnisse wird ebenfalls zu Generierung von Feedback genutzt.

Mit Hilfe der in Abschnitt 5 beschriebenen Komponente von CaseTrain können **Bildmarkierungsaufgaben** gestellt werden, bei welchen die Tätigkeit der Lernenden darin besteht, bestimmte Bereiche in einem Bild zu markieren. Diese Markierungen werden mit denen einer Musterlösung verglichen, wobei dieser Vergleich auf dem Konzept des Überdeckungsmaßes basiert. Bei Bildmarkierungsaufgaben handelt es sich nach Rütters Definition um halb-offene Aufgaben.

Bei den hier vorgestellten Trainingssystemen liegt der Fokus auf der Feedback-Generierung von Aufgaben, die im Kontext eines Übungsbetriebs oder zum freien Üben gestellt werden. Die Vermittlung von grundlegendem Wissen findet also nicht bei der Verwendung dieser Systeme, sondern beispielsweise in der Präsenzlehre statt. Sie sind also für den Einsatz im Sinne des Blended Learning konzipiert.

2. UML Klassendiagramme

In der Hochschulbildung wird in der Informatik und verwandten Studiengängen neben der Programmierausbildung auch objektorientierte Modellierung im Rahmen der Softwaretechnik bzw. des Software Engineering gelehrt. Als Werkzeug wird hier die weit verbreitete *Unified Modeling Language* (UML) [Rumbaugh et al., 2010][Balzert, 2005] der Version 2 verwendet. Studierende sollen dabei lernen, vor der Implementierung über ihre Programme nachzudenken und diese auf einem höheren Abstraktionsniveau zu skizzieren. In der UML können statische Konzepte durch Strukturdiagramme und dynamische Konzepte durch Verhaltensdiagramme abgebildet werden. Das wohl wichtigste Strukturdiagramm der UML ist das Klassendiagramm, in dem neben den Klassen auch deren Attribute, Operationen, Assoziationen und Generalisierungsstrukturen enthalten sind. UML Klassendiagramme sind sowohl zur Programmentwicklung als auch - aufgrund ihrer einfachen Abbildbarkeit auf Entity-Relationship-Diagramme - für den Datenbankentwurf beliebt und werden daher bereits in Informatik-Grundvorlesungen gelehrt. Dabei bedarf es einiger Erfahrung, um angemessene Klassendiagramme zu erstellen. Daher sind Übungen sehr wichtig in der Lehre. Der Zeitaufwand zur Korrektur von UML Klassendiagrammen durch Lehrpersonal ist beträchtlich, insbesondere dann, wenn ein lernförderliches Feedback gegeben werden soll.

Es wird in diesem Kapitel ein Verfahren zur automatischen Korrektur von Klassendiagrammen beschrieben und evaluiert, das auf Vergleichen der Klassendiagramme der Lernenden mit einer Musterlösung basiert, die Problemstellung dabei auf eine Reihe von linearen Zuordnungsproblemen abbildet und bis zu einem gewissen Grad tolerant ist. Zur Lösung dieser Zuordnungsprobleme müssen zwischen Elementen gleichen Typs (z. B. Klassen, Attribute, Operationen etc.) Ähnlichkeiten berechnet werden. Die entsprechenden Ähnlichkeitsfunktionen bzw. Ähnlichkeitsmaße, bei deren Berechnung unter anderem einige Vorverarbeitungstechniken des *Natural Language Processing* (NLP) verwendet werden, werden in diesem Kapitel ausführlich beschrieben.

2.1. Beschreibung des Aufgabentyps

Im Aufgabentext wird den Lernenden eine zu modellierende Domäne beschrieben. Dabei werden Eigenschaften von Entitäten und deren Beziehungen untereinander genannt. Die Aufgabe für den Lernenden besteht nun darin, ein Klassendiagramm zu erstellen, das konsistent zur im Aufgabentext beschriebenen Domäne ist.

Die Lernziele dabei sind:

- Identifizieren von Klassen, Attributen und Operationen aus der Domänenbeschreibung
- Erkennen von Vererbungsbeziehungen zwischen den Klassen anhand der Domänenbeschreibung
- Erkennen von Assoziationen zwischen Klassen inklusive der Kardinalitäten
- Erstellen eines syntaktisch korrekten Klassendiagramms

Es handelt sich bei diesem Aufgabentyp um eine offene, stark strukturierte Aufgabe (siehe Abschnitt 1.5.2): Lernende müssen hier eine Lösung konstruieren, nicht selektieren, zudem existiert keine eindeutige, korrekte Lösung der Aufgabe, da es im Allgemeinen mehrere Möglichkeiten gibt, ein syntaktisch korrektes Klassendiagramm zu erstellen, das konsistent zur beschriebenen Domäne ist.

2.1.1. Didaktische Variationen bei der Aufgabenstellung

Statt die Lernenden das Klassendiagramm von Grund auf selbst gestalten zu lassen, können ihnen auch Hilfestellungen gegeben werden, welche sich auch kombinieren lassen. Auf diese Weise können bei komplexen Domänen einfache Aufgaben gestellt werden.

2.1.1.1. (teilweise) Vorgabe von Klassen

Ein besonders wichtiger und für Anfänger nicht einfacher Schritt, ist die Identifikation der Klassen des Modells. Werden diese Klassen vorgegeben, beschränkt sich die Aufgabe der Studierenden auf das Modellieren der Beziehungen zwischen den Klassen und das Festlegen der Attribute und Operationen. Auch können die Klassen nur teilweise vorgegeben sein.

2.1.1.2. Vorgabe von Attributen und Operationen

Statt es den Lernenden zu überlassen, Attribute und Operationen anhand der textuellen Beschreibung der zu modellierenden Domäne selbst zu erstellen, können diese jeweils vorgegeben werden, allerdings ohne den entsprechenden Klassen zugewiesen zu sein. Diese Zuweisung der beispielsweise als Listen angezeigten Attributen und Operationen zu den Klassen wird dann von den Lernenden vorgenommen.

2.1.1.3. Konkrete Anweisungen zur Modellierung

Die Aufgabenstellung kann Anweisungen enthalten, wie bestimmte Sachverhalte der Domäne zu modellieren sind. Beispielsweise kann gefordert werden, dass eine bestimmte Entität als eigene Assoziations-Klasse oder eine Klasse als Interface modelliert werden soll.

2.2. Verwandte Arbeiten

In der Literatur gibt es bereits einige Ansätze zur automatischen Korrektur von UML Klassendiagrammen.

2.2.1. Automatische Auswertung durch Vergleich mit Musterlösungen

Die in den folgenden beiden Abschnitten beschriebenen Verfahren verwenden zur automatischen Auswertung von UML Klassendiagrammen einen Vergleich mit einer oder mehreren Musterlösungen.

2.2.1.1. UML Class Diagram Assessor

Einen Ansatz, der auf dem Vergleich zu einer Musterlösung basiert, stellen [Ali et al., 2007] mit dem *UML Class Diagram Assessor* (UCDA) vor.

Dabei werden die Klassendiagramme von Dozenten und Studierenden mit dem kommerziellen UML-Werkzeug *IBM Rational Rose*[1] erstellt. Dabei handelt es sich um ein umfangreiches Werkzeug, welches von Experten in der Softwareentwicklung eingesetzt wird.
Die von UCDA zur Überprüfung der Lernerlösungen anhand der Musterlösung verwendete Softwarekomponente besteht aus drei Modulen, die sequentiell ausgeführt werden, wobei jedes Modul den Studierenden im Falle von Fehlern Feedback gibt. Dabei wird das zweite (bzw. dritte) Modul nur dann ausgeführt, wenn das erste (bzw. zweite) keine Fehler liefert. Studierende können dabei ihr Diagramm iterativ verbessern.
Zuerst überprüft das *Class Structure Analysis Module* die Lernerlösung beispielsweise auf die korrekte Anzahl von Klassen, die korrekte Anzahl von Attributen innerhalb von Klassen oder die Korrektheit von Parametertypen. Das *Verification Process Module* generiert Fehlermeldungen bezüglich der Relationen zwischen den Klassen, also ob die Anzahl der Relationen insgesamt korrekt ist, ob die Anzahl der Relationen bestimmten Typs (Assoziation, Generalisierung, Aggregation, Komposition) korrekt ist und ob konkrete Relationen zwischen Klassen vorhanden sind. Das nur für die malaiische Sprache verfügbare *Language Checking Module* überprüft, ob bei der Benennung von Klassen und Attributen Substantive und bei der Benennung von Operationen Verben verwendet wurden.
Eine Möglichkeit, wie mehrere Musterlösungen verwendet werden können, wird nicht gezeigt. Auch bleibt unklar, wie festgestellt wird, ob eine bestimmte Klasse im Diagramm oder ein bestimmtes Attribut in einer Klasse vorhanden ist. Von einer Evaluation wird nicht berichtet.

[1] http://www-03.ibm.com/software/products/en/ratirosefami (August 2014)

2.2.1.2. ACME-DB

[Soler et al., 2010] stellen einen Ansatz vor, der in das webbasierte ACME-DB Framework integriert ist. Dabei handelt es sich um eine E-Learning-Plattform der Universität Girona (Spanien), welche in Kursen zu Datenbanken eingesetzt wird.
Es können zu jeder Aufgabe zur Erstellung von UML Klassendiagrammen mehrere Musterlösungen hinterlegt werden, wobei diese direkt in einer XML ähnlichen Struktur eingegeben werden. In einem webbasierten Editor können Lernende dann zu den gestellten Aufgaben Klassendiagramme eingeben und prüfen lassen. Nach einer Prüfung durch das Korrekturmodul werden den Lernenden entsprechende Fehler in ihrem Klassendiagramm in Textform mitgeteilt. Fehler sind beispielsweise eine unkorrekte Anzahl an Klassen, falsch benannte Klassen oder falsche Werte bei Kardinalitäten. Nach einer Prüfung kann die eigene Lösung weiter verbessert werden, so dass sich Studierende in einem iterativen Prozess der Musterlösung annähern.
Es bleibt unklar, wie das Korrekturmodul den Vergleich von Lernerlösung zu Musterlösungen durchführt, also beispielsweise wie erkannt wird, ob eine bestimmte Klasse aus der Musterlösung auch in der Lernerlösung vorkommt. Ebenfalls unklar bleibt, wie entschieden wird, welche Musterlösung bei einer Prüfung als Referenzlösung verwendet wird.
Das System wurde in einem Parallelgruppenvergleich mit 48 Studierenden evaluiert. Eine Gruppe bereitete sich mit 4 Aufgaben aus der ACME Lernplattform auf eine Prüfung vor, während die Studierenden der anderen Gruppe ermutigt wurden, die Aufgaben per Hand zu lösen und bei Fragen das Büro des Dozenten aufzusuchen. Es bleibt unklar, inwieweit die Studierenden der ersten Gruppe die Lernplattform tatsächlich benutzt haben und inwieweit Studierende der zweiten Gruppe die Aufgaben bearbeitet oder sich beraten lassen haben. Die Prüfung beinhaltete dabei eine Aufgabe, die ähnlich zu den zuvor gestellten Aufgaben war. Die Gruppe, welche die Lernplattform benutzen sollte, erzielte dabei im Durchschnitt leicht bessere Ergebnisse als die zweite Gruppe, doch waren die Unterschiede nicht signifikant. Die Eindrücke der Studierenden wurden insgesamt als positiv beschrieben, wurden jedoch nicht systematisch erfasst und ausgewertet.

2.2.2. Regelbasierte Korrektur

Einen Ansatz ohne explizite Musterlösungen verfolgen [Striewe und Goedicke, 2011]. Klassendiagramme werden hier als formale Graphen interpretiert, welche mit einer geeigneten Anfragesprache, einer sogenannten *graph query language*, untersucht werden können. Als Anfragesprache wird GReQL verwendet, mit welcher Anfragen gestellt werden können, die über die Existenz oder Nichtexistenz von Elementen anhand deren Typ oder dessen Eigenschaften Auskunft geben.
Um eine Aufgabe zu stellen, muss nun seitens des Aufgabenstellers neben der Domänenbeschreibung eine Reihe von Regeln mittels solcher Abfragen definiert werden. Dabei muss bei jeder Abfrage markiert werden, ob es sich um ein gewünschtes oder nicht gewünschtes Element handelt, damit bei der automatischen Korrektur entsprechende Feh-

lermeldungen gegeben werden können. Die Regeln können dabei logisch kombiniert werden, so dass beispielsweise die Akzeptanz alternativer Schreibweisen abgebildet werden kann. Es ist möglich, mit diesen Regeln auch stilistische Eigenschaften der Lernerlösung zu prüfen, beispielsweise ob die Namen aller vorkommenden Klassen mit Großbuchstaben beginnen. Die folgende Regel prüft auf das Vorkommen einer Klasse mit dem Namen „Tariff“, welche ein Attribut beinhalten muss, welches „Name“ oder „Identifier“ heißt:

```
from x : V{Class}, y : V{Property}
with x --> y and x.name="Tariff" and
     (capitalizeFirst(y.name)="Name" or
      capitalizeFirst(y.name)="Identifier")
report x.name, y.name end
```

Da Dozenten, die Aufgaben zu UML Klassendiagrammen stellen, GReQL im Allgemeinen wohl nicht beherrschen, müssen diese Regeln also von entsprechenden Experten eingegeben werden. Es wird hier nur auf die Identität zu den entsprechenden Namen geprüft, alternative Schreibweisen, die beispielsweise durch Flexion oder Tippfehler entstehen, müssten in den Regeln explizit angegeben werden. Für eine Beispielaufgabe mussten 17 solche Regeln definiert werden. 5 davon sind allerdings allgemeine Regeln, die auch in anderen Aufgaben weiter verwendet werden können, so dass speziell für diese Aufgabe 12 Regeln definiert werden mussten.

Für die Auswertung, also die Berechnung der Punktzahl für eine Lernerlösung, gibt es nun zwei Strategien. Entweder wird von 0 Punkten ausgegangen und Studierende erhalten Punkte für jede Regel, die nach einem gewünschten Element sucht und dieses auch findet (pessimistischer Ansatz). Oder es wird von der maximalen Punktzahl ausgegangen und Studierenden werden für jede Regel Punkte abgezogen, die nach einem gewünschten Element sucht und dieses nicht findet (optimistischer Ansatz). Der Punktwert muss bei der Definition der entsprechenden Regel hinterlegt werden.

In einem Experiment, bei welchem der pessimistische Ansatz verwendet wurde, wurde sechs Gruppen von Studierenden eine Aufgabe vorgelegt, die diese in Teamarbeit lösten. Die automatischen Bewertungen der Lernerlösungen reichten von 67 bis 91 von 100 Punkten. Diese Bewertungen lagen im gleichen Bereich der Bewertungen, die ein menschlicher Korrektor vorgenommen hatte, dem die Regeln zur automatischen Bewertung nicht bekannt waren. Über die Korrelation der Bewertungen der beiden Methoden wird allerdings keine Aussage gemacht. Es zeigte sich außerdem, dass die definierten Regeln nicht alle Aspekte abdecken konnten, die bei einer manuellen Bewertung betrachtet würden. Dies wird nicht als Problem der Technik beschrieben, sondern als Problem der Tatsache, dass die Regeln geschrieben werden, ohne die Lernerlösungen zu kennen. Es ist also nötig, die Regeln in einem iterativen Prozess nach der Bewertung einiger Lernerlösungen anzupassen.

Unter der Voraussetzung, dass eine praxiserprobte Menge an allgemeinen Regeln besteht und dass Dozenten die Fähigkeit besitzen, in akzeptabler Zeit gute Regeln zu erstellen, verspricht dieser Ansatz gute Ergebnisse. Positiv hervorzuheben ist, dass anhand der Anfrageergebnisse ein eindeutiges Feedback, beispielsweise in natürlichsprachiger Form,

generiert werden kann, in welchem mitgeteilt wird, welche Regeln verletzt wurden. Das Fehlen einer Musterlösung im Feedback ist allerdings durchaus als Nachteil zu sehen. Ebenfalls bleibt die Problematik verschiedener Schreibweisen von Elementen ohne deren explizite Angabe ungelöst, wobei die Autoren andeuten, dass es eine Möglichkeit gibt, GReQL um entsprechende Funktionen zu erweitern.

2.2.3. Zusammenfassung

Es wurden einige Ansätze aus der Literatur zur automatischen Bewertung von UML Klassendiagrammen beschrieben. Keines dieser Verfahren scheint dabei insofern tolerant zu sein, als dass auch von der Musterlösung abweichend benannte Elemente der Lernerlösung (zumindest teilweise) akzeptiert werden. Dass dazu ein Bedarf besteht, belegt ein Experiment, welches zeigt, dass über 80 %von etwa 500 von Studierenden frei gezeichnete Klassendiagramm Klassen enthielten, deren Namen zu keiner Klasse aus der zugehörigen Musterlösung identisch waren (siehe Abschnitt 2.6.2.2). Bei Attributen ist der Anteil ähnlich hoch.

Außerdem bleibt unklar, wie bei ACME-DB die Referenzlösung aus den hinterlegten Musterlösungen gewählt wird. Als problematisch anzusehen ist die komplex erscheinende Eingabe der Regeln des von [Striewe und Goedicke, 2011] vorgestellten Ansatzes, die eine Hürde für Dozenten darstellen kann, ein entsprechendes Trainingssystem einzusetzen. Auch die Eingabe der Klassendiagramme über ein kommerzielles Werkzeug erscheint problematisch, da zu einem praktischen Einsatz entsprechende Lizenzen kostenpflichtig erworben werden müssen.

Auch mit dem von [Hoffmann et al., 2008] beschriebenen System DUESIE lassen sich Aufgaben zu Klassendiagrammen stellen. Dabei wird aber nicht genauer auf das Verfahren zur Auswertung der Klassendiagramme eingegangen.

2.3. Feedback-Generierung zu UML Klassendiagrammen durch Berechnung des Überdeckungsmaßes zur Musterlösung

Im Folgenden stellen wir einen Ansatz zur automatischen Auswertung von und Feedback-Generierung zu frei gezeichneten UML Klassendiagrammen vor, der auf der in Abschnitt 1.6.1 beschriebenen Berechnung eines Überdeckungsmaßes von mehreren Musterlösungen durch Lernerlösungen basiert.
Durch die Verwendung einiger Vorverarbeitungstechniken des NLP ist die Korrektur zu einem gewissen Maße tolerant, was die Schreibweise von Namen der in den Klassendiagrammen enthaltenen Elemente betrifft. Die Auswahl der geeigneten Referenzlösung aus den hinterlegten Musterlösungen ergibt sich dabei trivial: Es wird diejenige Musterlösung gewählt, zu der die Lernerlösung die größte Überdeckung aufweist.
Dabei wird Feedback der Formen KP, KCR und EF generiert (siehe Abschnitt 1.4 auf

Seite 4). Das Feedback der Form EF hat tabellarische Form mit textuellen Erläuterungen. Für Autoren der Aufgaben besteht nur sehr wenig Aufwand, da außer der Erstellung einer oder mehrerer Musterlösungen keine weitere inhaltliche Arbeit zu leisten ist.

2.3.1. Überdeckungsmaß für Klassendiagramme

Das Überdeckungsmaß $U_K(A, B)$ zweier Klassendiagramme A und B gibt an, inwiefern ein Klassendiagramm A von einem Klassendiagramm B „überdeckt“ wird. Es handelt sich dabei um eine andere Funktion als das in Abschnitt 1.6.1 beschriebene Überdeckungsmaß von Mengen, jedoch ist das Konzept von diesem abgeleitet, so dass der gleiche Name verwendet wird. Zur Berechnung des Überdeckungsmaßes von Klassendiagrammen werden diese in Komponenten zerlegt, auf deren Basis sie verglichen werden. Die Komponenten eines Klassendiagramms sind jeweils stets Element genau einer der folgenden Mengen:

- K Menge aller Klassen
- V Menge aller Vererbungsbeziehungen
- A Menge aller Assoziationen
- M Menge aller Multiplizitäten (von Assoziationen)[2]
- T Menge aller Attribute
- O Menge aller Operationen

K enthält also nicht nur die Klassen eines bestimmten Klassendiagramms, sondern alle möglichen Klassen aller denkbaren Klassendiagramme. Für die anderen Mengen gilt dies analog.

Wir betrachten ein Klassendiagramm D nun als ein 6-Tupel $(K_D, V_D, A_D, M_D, T_D, O_D)$. Dabei ist:

- K_D: Menge aller in D enthaltenen Klassen
- V_D: Menge aller in D enthaltenen Vererbungsbeziehungen
- A_D: Menge aller in D enthaltenen Assoziationen
- M_D: Menge aller in D enthaltenen Multiplizitäten ($|M_D| = |A_D|$)
- T_D: Menge aller in D enthaltenen Attribute
- O_D: Menge aller in D enthaltenen Operationen

Dabei gilt $\forall k \in K_D : k \in K$. Seien A und B zwei Klassendiagramme und seien E_A und E_B jeweil Objekte gleichen Typs aus den 6-Tupeln, die A und B definieren (also beispielsweise die Klassen aus A und B, also K_A und K_B). Es wird nun für jedes Paar dieser Mengen ein Zuordnungsproblem gelöst, wobei analog zu obiger Beschreibung bei

[2] Eine Assoziation hat im Allgemeinen zwei Multiplizitäten, die in diesem Verfahren zu einem einzigen Wert zusammengefasst werden.

der Berechnung des Überdeckungsmaßes vorgegangen wird und je Typ ein **Ähnlichkeitsmaß** definiert ist, also eine Funktion S, die für zwei Elemente gleichen Typs eine Ähnlichkeit berechnet:

$$\mathbb{R}^{01} = \{r \mid r \in \mathbb{R}, 0 \leq r \leq 1\} \tag{2.1}$$
$$S_K : K \times K \mapsto \mathbb{R}^{01} \tag{2.2}$$
$$S_V : V \times V \mapsto \mathbb{R}^{01} \tag{2.3}$$
$$S_A : A \times A \mapsto \mathbb{R}^{01} \tag{2.4}$$
$$S_M : M \times M \mapsto \mathbb{R}^{01} \tag{2.5}$$
$$S_T : T \times T \mapsto \mathbb{R}^{01} \tag{2.6}$$
$$S_O : O \times O \mapsto \mathbb{R}^{01} \tag{2.7}$$

Wenn alle sechs Zuordnungsprobleme gelöst sind, ist jedem Element aus A ein Element aus B (oder ein Dummy-Element) zugeordnet. Es ist nun für jeden Element-Typ eine Bewertungsfunktion B_E definiert, die für ein Paar (e_A, e_B) dieser Zuordnung ein **Bewertungsmaß** zwischen 0 und 1 berechnet. Das Bewertungsmaß gibt an, wie gut e_A von e_B überdeckt wird.

$$\mathbb{R}^{01} = \{r \mid r \in \mathbb{R}, 0 \leq r \leq 1\} \tag{2.8}$$
$$B_K : K \times K \mapsto \mathbb{R}^{01} \tag{2.9}$$
$$B_V : V \times V \mapsto \mathbb{R}^{01} \tag{2.10}$$
$$B_A : A \times A \mapsto \mathbb{R}^{01} \tag{2.11}$$
$$B_M : M \times M \mapsto \mathbb{R}^{01} \tag{2.12}$$
$$B_T : T \times T \mapsto \mathbb{R}^{01} \tag{2.13}$$
$$B_O : O \times O \mapsto \mathbb{R}^{01} \tag{2.14}$$

Ist ein Element eines Paares ein Dummy-Element, so ist das Bewertungsmaß stets gleich 0. Bei einigen Element-Typen entspricht die Bewertungsfunktion B_E der Ähnlichkeitsfunktion S_E, welche zur Lösung des Zuordnungsproblems verwendet wurde. Bei bestimmten Element-Typen ist diese weitere Funktion zur Bewertung deshalb nötig, weil die Berechnung der Ähnlichkeitsfunktion eine andere Intention hat als die der Bewertungsfunktion. Beispielsweise dient die Ähnlichkeitsfunktion S_K dazu, bei der Lösung des entsprechenden Zuordnungsproblems eine Zuordnung zwischen den Klassen zu finden, die die gleiche Entität der Domäne modellieren, wobei der Typ der Klasse (konkrete Klasse, abstrakte Klasse oder Schnittstelle) nicht berücksichtigt wird. Die Bewertungsfunktion gibt dann an, wie gut die Klasse aus B die Klasse aus A überdeckt, wobei der Typ der Klasse in der Bewertungsfunktion mit einbezogen wird.

Zur Berechnung des Überdeckungsmaßes von A und B werden nun die Bewertungsmaße aller Element-Paare addiert, die keine Dummy-Elemente beinhalten, und mittels Division durch die Mächtigkeit von A normiert, so dass ein Überdeckungsmaß zwischen 0 und 1 resultiert. Zusätzlich wird für jeden Element-Typ noch eine Gewichtung $(G_K \dots G_O)$ zwischen 0 und 1 eingeführt, so dass beispielsweise die Überdeckung einer Klasse höher gewichtet werden kann als die eines Attributs.

$$U(A,B) = \frac{\sum_{k_A \in K_A} G_K \cdot B_K(k_A, Z_K(k_A)) + \dots + \sum_{o_A \in O_A} G_O \cdot B_O(o_A, Z(o_A))}{G_K \cdot |K_A| + \dots + G_O \cdot |O_A|} \quad (2.15)$$

Dabei sind $Z_K \dots Z_O$ die Lösungen der jeweiligen Zuordnungsprobleme, also die Bijektionen, die die jeweilige Nutzenfunktion maximieren. $(k_A, Z_K(k_a))$ ist also beispielsweise ein Paar einander zugeordneter Klassen.

Die in den folgenden Abschnitten vorgestellten Ähnlichkeits- und Bewertungsmaße haben also eine jeweils unterschiedliche Funktion:

- Das **Ähnlichkeitsmaß** dient jeweils als Gewichtsfunktion bei den entsprechenden linearen Zuordnungsproblemen.
- Das **Bewertungsmaß** wird erst anschließend verwendet. Sind die Elemente durch das Lösen des Zuordnungsproblems einander zugeordnet, wird mittels der Bewertungsfunktion für jedes dieser Paare ein Bewertungsmaß berechnet, welches für das Feedback der Form KP, also die quantitative Rückmeldung, verwendet wird.

Bei der Berechnung der Ähnlichkeits- und Bewertungsfunktionen werden die folgenden Eigenschaften der Elemente berücksichtigt:

- Klassen
  - Name
  - Attribute
  - Operationen
  - Typ (konkrete Klasse, abstrakte Klasse, Schnittstelle, enum)
- Vererbungsbeziehungen
  - beteiligte Klassen (Superklasse und Subklasse)
- Assoziationen
  - beteiligte Klassen
- Multiplizitäten
  - zugehörige Assoziation
  - Zahlenwerte der Multiplizität
- Attribute
  - zugehörige Klasse

  - Name
  - Datentyp
- Operationen
  - zugehörige Klasse
  - Name
  - Ausgabeparameter (Datentyp)
  - Eingabeparemeter (Name und Datentyp)

Die Ähnlichkeitsmaße zwischen Elementen werden je nach Typ verschieden berechnet. Bei Elementen einiger Typen (Klassen, Attribute, Operationen) basiert die Berechnung der Ähnlichkeit auf der Ähnlichkeit von deren Namen, also Zeichenketten.

2.3.2. Ähnlichkeit von Zeichenketten

Bei Klassendiagrammen sind vorkommende Zeichenketten in der Regel natürlichsprachige Wörter oder aus natürlichsprachigen Wörtern zusammengesetzt, so dass „Wort“ und „Zeichenkette“ im Folgenden synonym verwendet werden. Vorerst wird davon ausgegangen, dass eine Zeichenkette nur ein Wort beinhaltet.

Die Ähnlichkeit zweier Zeichenketten ist 1, wenn die Zeichenketten identisch sind. Der Vergleich erfolgt dabei ohne Beachtung von Groß- und Kleinschreibung, so dass beispielsweise „Bibliothek“ und „bibliotheK“ als identisch gelten.

Sind die Zeichenketten nicht identisch, werden beide Zeichenketten als natürlichsprachige Wörter behandelt und mittels eines sogenannten „Stemmers“ [Lovins, 1968] auf ihre Grundform reduziert. So werden Substantive stets in die erste Person, Nominativ, Singular umgewandelt. Dieser Vorgang wird auch als „Stemming“ bezeichnet. Sind die Grundformen (ohne Beachtung der Groß- und Kleinschreibung) beider Worte nun identisch, ist die Ähnlichkeit nahe an, aber kleiner als 1. Der konkrete Wert ist statisch und bei der Implementierung als Parameter zu wählen. Die Ähnlichkeit sollte deshalb nicht gleich 1 sein, damit zwei Zeichenketten A und B, die identisch sind, eine höhere Ähnlichkeit zugewiesen wird, als zwei Zeichenketten A und C, bei denen lediglich die Grundform identisch ist. Durch die Grundformreduktion können bei Klassen, Attributen, Operationen und Parametern solche Fälle behandelt werden, bei denen Lernende Namen von der Musterlösung abweichend konjugiert oder dekliniert haben.

Sind auch die Grundformen nicht identisch, wird die sogenannte Levenshtein-Distanz [Levenshtein, 1966] zwischen den Zeichenketten berechnet. Diese Distanz gibt an, wie viele Änderungsoperationen notwendig sind, um eine Zeichenkette in die andere zu überführen. Gültigen Operationen dabei sind: Einfügen, Löschen oder Ersetzen von Zeichen. Ist die Levenshtein-Distanz kleiner oder gleich einem zuvor definierten Schwellwert S, wird aus der Levenshtein-Distanz eine Ähnlichkeit S_L berechnet, indem durch Division mit der maximalen Levenshtein-Distanz der Zeichenketten normiert und dieses Ergebnis von 1 subtrahiert wird. Die maximale Levenshtein-Distanz zweier Zeichenkette ist die

Länge der längeren Zeichenkette, also die Anzahl der enthaltenen Zeichen. Seien A und B zwei Zeichenketten, $L(A)$ und $L(B)$ deren Längen und $D_L(A,B)$ deren Levenshtein-Distanz, dann ist:

$$S_L(A,B) = \begin{cases} 1 - \frac{D_L(A,B)}{max(L(A),L(B))} & D_L(A,B) \leq S \\ 0 & sonst \end{cases} \tag{2.16}$$

Beispiel: Die Levenshtein-Distanz der Zeichenketten „bibliothek“ und „bilbiotek“ beträgt 3. Die maximale Levenshtein Distanz beträgt 10. Die resultierende Ähnlichkeit ist also $\frac{7}{10} = 0{,}7$.

Die Berücksichtigung der Levenshtein-Distanz ermöglicht, eine Ähnlichkeit zwischen Zeichenketten zu erkennen, die mit Tippfehlern behaftet sind oder deren Grundformreduktion nicht korrekt funktionierte. Ein Schwellwert für die Levenshtein-Distanz muss deshalb gewählt werden, damit Wörtern, die bezüglich ihrer Zeichen und deren Reihenfolge ähnlich sind, aber verschiedene Konzepte beschreiben, eine Ähnlichkeit gleich 0 zugewiesen wird. So haben die Wörter „Buch“ und „Bus“ eine Levenshtein-Distanz von nur 2, sind aber semantisch sehr verschieden.

Ist die Levenshtein-Distanz größer als der Schwellwert, werden in einem angebundenen elektronischen Wörterbuch auf Basis der Grundformen der Wörter jeweils Synonyme gesucht. Als Synonymität wird in der Sprachwissenschaft die Bedeutungsgleichheit zweier Worte bezeichnet. Es werden nun die Ähnlichkeiten zwischen dem Wort A und den Synonymen von Wort B (also deren Zeichenketten) analog zu eben genanntem Verfahren mit Grundformreduktion und Abgleich der Levenshtein-Distanz berechnet, wobei es auch für die durch Synonyme errechnete Ähnlichkeit eine obere Schranke nahe 1 gibt, damit Synonymen stets eine kleinere Ähnlichkeit als identischen Wörtern zugewiesen wird. Der gleiche Vorgang wird auch mit vertauschter Reihenfolge von A und B durchgeführt. Die Ähnlichkeit ist dann der maximale Wert aller Ähnlichkeiten von A zu den Synonymen von B bzw. von B zu den Synonymen von A.

In der Praxis kommt es häufig vor, dass Namen von Elementen (insbesondere von Operationen) aus mehreren Wörtern bestehen, die aus syntaktischen Gründen[3] ohne Leerzeichen aneinandergereiht werden, wobei die ersten Buchstaben der Wörter (mit Ausnahme des ersten) groß, also als Binnenmajuskel, geschrieben werden.[4] Diese Schreibweise wird im Kontext von Programmiersprachen auch als „CamelCase“ bezeichnet. Die Zeichenkette kann nun anhand der Großbuchstaben in mehrere sogenannte „Token“ aufgeteilt werden, die zusammen ein „Textsegment“ bilden. Die Ähnlichkeit der Zeichenketten ist

[3] In den meisten Programmiersprachen dürfen Bezeichner keine Leerzeichen enthalten. Eine Ausnahme ist beispielsweise FORTRAN.

[4] Es gibt auch weitere Möglichkeiten, die Wortgrenzen zu markieren, beispielsweise durch die Verwendung eines Trennzeichens wie „_“. Dies muss in der Implementierung des Verfahrens entsprechend berücksichtigt werden.

nun gleich der Ähnlichkeit dieser beiden Textsegmente. Textsegmente werden dabei als Mengen von Token behandelt, so dass die Reihenfolge der Token keine Bedeutung hat. Dies wird auch als das „bag-of-words“-Prinzip bezeichnet.

2.3.3. Ähnlichkeit von Textsegmenten

Die Berechnung der Ähnlichkeit zweier Textsegmente geschieht nach dem oben beschriebenen Verfahren zur Berechnung der Ähnlichkeit von Mengen (siehe Abschnitt 1.6.1.1 auf Seite 14).

Beispiel: Gegeben seien die Textsegmente „buch leihen“ und „buch exemplar ausleihen“, und seien die Wörter „leihen“ und „ausleihen“ im angebundenen Wörterbuch als Synonyme hinterlegt. Dann gilt: $A = \{buch, leihen\}$ und $B = \{buch, exemplar, ausleihen\}$, das Überdeckungsmaß $U(A, B) = 1$ und das Überdeckungsmaß $U(B, A) = \frac{2}{3}$. Es resultiert eine Ähnlichkeit von $S(A, B) = \frac{5}{6}$.

2.3.4. Ähnlichkeit von Datentypen

Bei der Berechnung der Ähnlichkeits- und Bewertungsmaße von Attributen werden auch deren Datentypen, bei Operationen die Rückgabetypen sowie die Datentypen der Eingabeparemeter berücksichtigt. Die Ähnlichkeit zweier Datentypen wird dabei wie folgt berechnet. Sind die Namen der Datentypen ohne Berücksichtigung von Groß- und Kleinschreibung identisch, so ist die Ähnlichkeit gleich 1.

Sind sie nicht identisch, wird die Kategorie des Datentyps betrachtet. Welche Kategorien im Kontext einer Aufgabe existieren, hängt davon ab, welche Datentypen im Klassendiagramm eingegeben werden dürfen, wobei dies eine didaktische Entscheidung ist. In der UML sind die folgenden vier Datentypen vordefiniert: `Boolean` (Wahrheitswerte), `String` (Zeichenketten), `Integer` (ganze Zahlen) und `UnlimitedNatural` (natürliche Zahlen). Werden nun per Konvention (beispielsweise in der Aufgabenstellung) weitere Datentypen zugelassen und sind diese dem System zur automatischen Auswertung bekannt, können sie zusammen mit den vier von der UML vorgegebenen Datentypen zu Kategorien zusammengefasst werden, beispielsweise in numerische Datentypen (`Integer`, `UnlimitedNatural`, `int`, `Double`, `BigDecimal`,...) oder Zeichen-Datentypen (`String`, `char`, `Character`,...). Für diese Kategorien kann nun paarweise eine Ähnlichkeit zwischen 0 und 1 festgelegt werden. Die Kategorien können auch hierarchisch angelegt sein, beispielsweise lassen sich numerische Datentypen noch in Datentypen zu Darstellung von ganzen Zahlen einerseits und Gleitkommazahlen andererseits aufteilen. Sind die zu vergleichenden Datentypen nicht in den Kategorien enthalten, wird überprüft, ob beide im jeweiligen Klassendiagramm mittels einer Klasse selbst modelliert wurden, also ob der Name des Datentyps identisch zum Namen einer im selben Klassendiagramm definierten Klasse ist. Ist dies der Fall und wurden die beiden Klassen bei der Berechnung

des Überdeckungsmaßes der Klassen einander zugeordnet, so ist die Ähnlichkeit gleich 1. Ansonsten ist die Ähnlichkeit gleich 0.

2.3.5. Ähnlichkeit und Bewertung von Komponenten von Klassendiagrammen

2.3.5.1. Ähnlichkeit und Bewertung von Klassen

Um also ein Überdeckungsmaß zwischen zwei Klassendiagrammen berechnen zu können (siehe Abschnitt 2.3.1), müssen die Überdeckungsmaße deren Mengen von Komponenten bestimmt werden. Die wohl wichtigsten Komponenten in Klassendiagrammen sind eben deren Klassen. Der Berechnung des Überdeckungsmaßes der Klassen (und damit auch der Berechnung der Ähnlichkeit von Klassen) kommt dabei eine besondere Bedeutung zu. Durch das Lösen des bei der Berechnung definierten Zuordnungsproblems ergibt sich eine Zuordnung der Klassen der Musterlösung zu den Klassen der Lernerlösung. Für die Berechnung der Ähnlichkeiten weiterer Elemente ist diese Zuordnung von herausragender Bedeutung (siehe folgende Abschnitte). Werden hier falsche Zuordnungen vorgenommen, wird also nicht erkannt, dass eine Klasse der Lernerlösung einer Klasse der Musterlösung entspricht, können auch die mit der Klasse zusammenhängenden weiteren Elemente nicht einander zugeordnet und entsprechend bewertet werden.

Bei der Lösung des Zuordnungsproblems zur Berechnung des Überdeckungsmaßes der Klassen wird (im Gegensatz zur Berechnung bei anderen Typen) nach einem zweistufigen Modell vorgegangen, bei dem das Zuordnungsproblem zuerst mit einer Ähnlichkeitsfunktion S_N gelöst wird. Für die Klassen beider Mengen, für die in dieser ersten Stufe keine Zuordnung mit einem Gewicht größer als 0 gefunden wurde, wird erneut ein Zuordnungsproblem gelöst, wobei nun eine Ähnlichkeitsfunktion S_{AO} verwendet wird. Anschließend werden die Zuordnung aus beiden Lösungen zusammengefügt: Definitions- und Zielmengen beider die Nutzenfunktion maximierenden Bijektionen werden jeweils vereint und die Zuordnungen werden beibehalten, sodass sich erneut eine Bijektion ergibt. Seien A und B zwei Klassen verschiedener Klassendiagramme.

- Die Ähnlichkeitsfunktion $\boldsymbol{S_N(A,B)}$ entspricht der Ähnlichkeitsfunktion der Namen der Klassen, also deren Zeichenketten bzw. Textsegmente (siehe Abschnitt 2.3.2 und 2.3.3).
- Die Ähnlichkeitsfunktion $\boldsymbol{S_{AO}(A,B)}$ basiert auf der Berechnung der Überdeckungsmaße der Namen (also Zeichenketten) der in den Klassen vorkommenden Attribute und Operationen. Sei A_A bzw. A_B die Menge der Namen aller Attribute der Klasse A bzw. B und O_A bzw. O_B die Menge der Namen aller Operationen der Klasse A bzw. B. Die Ähnlichkeit der Klassen wird nun wie folgt berechnet, wobei U_N das Überdeckungsmaß zweier Mengen von Namen (also Zeichenketten) ist:

$$S_{AO}(A,B) = \frac{U_N(A_A, A_B) + U_N(O_A, O_B)}{2} \tag{2.17}$$

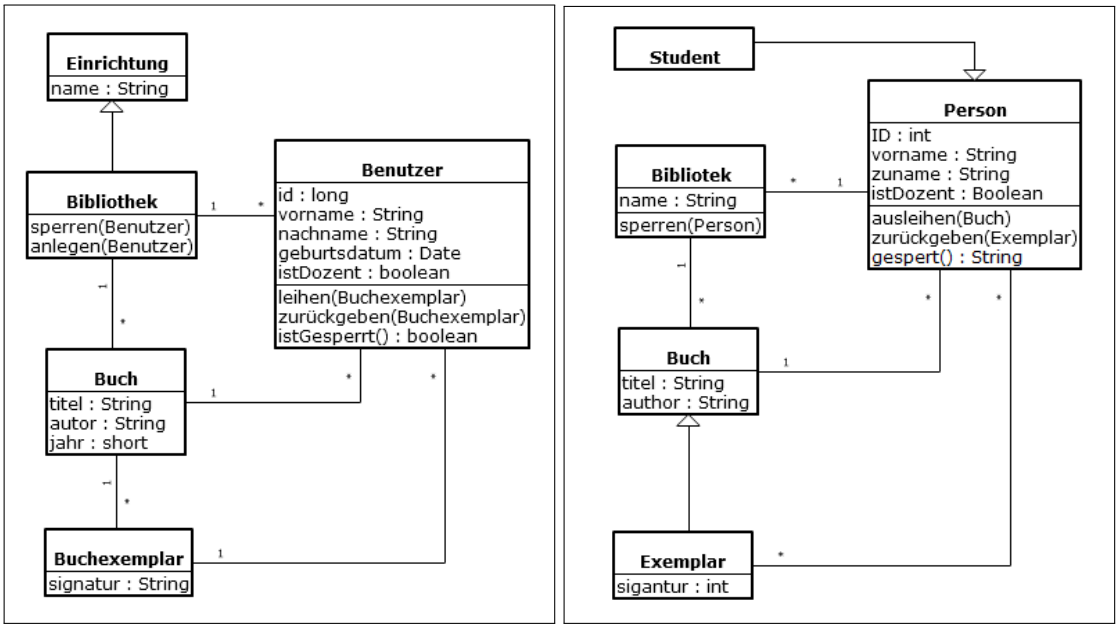


Abbildung 2.1.: UML Klassendiagramm: Beispiel einer Musterlösung (links) und Lernerlösung (rechts)

Dieses zweistufige Modell dient dazu, Klassen einander zuzuordnen, bei denen die modellierte Entität identisch, die Klassennamen sich jedoch nicht ähnlich genug sind, um in der ersten Stufe einander zugeordnet zu werden, beispielsweise wenn zu viele Tippfehler gemacht oder Synonyme nicht als solche erkannt wurden. Dies Zuordnung kann nun doch vorgenommen werden, wenn die enthaltenen Attribute und Operationen ähnlich genug sind.

Das Bewertungsmaß B_K zweier einander zugeordneter Klassen ist dabei stets 1, sofern beide Klassen den gleichen Typ (konkrete Klasse, abstrakte Klasse oder Schnittstelle) haben. Ist der Typ verschieden, so liegt das Bewertungsmaß zwischen 0 und 1, wobei dieser konstante Wert zuvor festzulegen ist. Die Namen einander zugeordneter Klassen haben für die Bewertung also keine Relevanz.

Beispiel: Es soll innerhalb der Domäne einer Universität eine Bibliothek mit Benutzern und Büchern modelliert werden. Abbildung 2.1 zeigt links eine Muster- und rechts eine Lernerlösung. Die Namen der Klassen beider Klassendiagramme seien im angebundenen Wörterbuch nicht als Synonyme hinterlegt.

In der ersten Stufe, bei der die Ähnlichkeit der Klassen S_N nur über deren Namen berechnet wird, werden die Klassen (*Buch*, *Buch*) und trotz des Schreibfehlers auch die Klassen (*Bibliothek*, *Bibliotek*) einander zugeordnet. Weitere Zuordnungen finden in dieser ersten Stufe nicht statt, da die Ähnlichkeiten der Namen der sonstigen Paare jeweils 0 sind. Auch die Ähnlichkeit zwischen *Buchexemplar* und *Exemplar* ist gleich 0, da

hier keine Binnenmajuskel verwendet wurde und die Wortgrenzen somit nicht erkannt werden konnten. In der zweiten Stufe werden aus der Musterlösung nur die Klassen *Einrichtung*, *Benutzer* und *Buchexemplar*, aus der Lernerlösung nur die Klassen *Student*, *Person* und *Exemplar* berücksichtigt. Hier ergibt sich bei der Lösung des Zuordnungsproblems und Verwendung der Ähnlichkeitsfunktion S_{AO}, dass die Namen der Attribute und Methoden der Klassen *Benutzer* und *Person* sehr ähnlich sind. Gleiches gilt für die Klassen *Buchexemplar* und *Exemplar*. Die Klassen werden also jeweils einander zugeordnet, so dass sich folgende Zuordnung ergibt: {(*Buch*, *Buch*), (*Bibliothek*, *Bibliotek*), (*Benutzer*, *Person*), (*Buchexemplar*, *Exemplar*)}. In den folgenden Schritten können nun auch Assoziationen, Attribute und Methoden der im zweiten Schritt zugeordneten Klassen positiv bewertet werden, was bei einer nicht vorgenommenen Zuordnung der beiden Klassen nicht möglich gewesen wäre. Die Klassen *Einrichtung* und *Student* bleiben ohne Zuordnung. Das Bewertungsmaß aller einander zugeordneter Klassen ist gleich 1.

2.3.5.2. Ähnlichkeit und Bewertung von Vererbungsbeziehungen

Die Ähnlichkeit S_V von Vererbungsbeziehungen kann nur im Kontext einer zuvor vorgenommenen Zuordnung von Klassen berechnet werden. Falls die an der Vererbungsbeziehung beteiligten Klassen nicht jeweils eine Zuordnung im anderen Klassendiagramm haben, ist die Ähnlichkeit 0. Haben sie eine Zuordnung und ist die Richtung der Vererbung identisch, ist die Ähnlichkeit 1. Haben sie eine Zuordnung bei verschiedener Richtung, ist die Ähnlichkeit gleich einem zuvor definierten Wert.

Das Bewertungsmaß B_V zweier einander zugeordneter Vererbungsbeziehungen entspricht deren Ähnlichkeitsmaß.

Beispiel: Das Ähnlichkeits- und Bewertungsmaß der beiden Vererbungsbeziehungen aus der Lernerlösung zur Vererbungsbeziehung aus der Musterlösung aus Abbildung 2.1 sind gleich 0.

2.3.5.3. Ähnlichkeit und Bewertung von Assoziationen

Die Ähnlichkeit S_A von Assoziationen ist stets 0 oder 1 und kann ebenfalls nur im Kontext einer zuvor vorgenommenen Zuordnung von Klassen berechnet werden. Falls die assoziierten Klassen jeweils eine Zuordnung im anderen Klassendiagramm haben, ist die Ähnlichkeit 1, ansonsten 0. Im hier vorgestellten Verfahren werden die Assoziationstypen Komposition und Aggregation nicht gesondert betrachtet, was aber leicht ergänzt werden kann.

Das Bewertungsmaß B_A zweier einander zugeordneter Assoziationen ist stets 1.

Beispiel: Ähnlichkeits- und Bewertungsmaß der Assoziationen *Buchexemplar-Benutzer* und *Exemplar-Person* aus Abbildung 2.1 sind gleich 1.

2.3.5.4. Ähnlichkeit und Bewertung von Multiplizitäten

Die Ähnlichkeit S_M von Multiplizitäten ist genau dann gleich 1, wenn die Ähnlichkeit der korrespondierenden Assoziationen gleich 1 ist, ansonsten ist sie gleich 0.

Das Bewertungsmaß B_M wird anhand der Werte der Multiplizitäten berechnet. Beim hier vorgestellten Verfahren gibt es eine Beschränkung auf Assoziationen mit den Multiplizitäten 1:1, 1:*, *:1 und *:*. Andere Multiplizitäten werden wie folgt abgebildet: $[0 \dots 1] \mapsto 1$ und $[2 \dots *] \mapsto *$. Es werden die Werte der Multiplizitäten verglichen und ein entsprechendes Bewertungsmaß wird bestimmt, wobei die Bewertungen der Werte paarweise wie folgt definiert ist:

	1:1	1:*	*:1	*:*
1:1	1	t	t	0
1:*	t	1	0	t
*:1	t	0	1	t
:	0	t	t	1

Eine Verfeinerung dieses Bewertungsmaßes ist natürlich möglich.

Dabei ist t das Bewertungsmaß für teilweise korrekte Multiplizitäten. Der Wert muss zuvor festgelegt werden, wobei $0 \leq t \leq 1$ sein muss. Bei geeigneter Wahl von t können teilweise korrekte Multiplizitäten also auch als gänzlich unkorrekt oder völlig korrekt bewertet werden. Die Bewertungsfunktion von Multiplizitäten ist kommutativ.

Beispiel: Das Ähnlichkeitsmaß der Multiplizitäten der korrespondierenden Assoziationen *Buchexemplar-Benutzer* und *Exemplar-Person* aus Abbildung 2.1 ist gleich 1. Das Bewertungsmaß ist t, da 1:* mit 1:1 verglichen wird. Wenn $t = 0{,}5$ angenommen wird, ist deren Bewertungsmaß also 0,5.

2.3.5.5. Ähnlichkeit und Bewertung von Attributen

Auch die Ähnlichkeit S_T von Attributen kann nur im Kontext einer zuvor vorgenommenen Zuordnung von Klassen berechnet werden. Sind die Klassen der beiden Attribute nicht einander zugeordnet worden, ist die Ähnlichkeit gleich 0. Andernfalls wird die Ähnlichkeit der Namen beider Attribute (siehe Abschnitt 2.3.2) und die Ähnlichkeit der Datentypen beider Attribute berechnet. Diese Ähnlichkeiten ergeben nun die Ähnlichkeit der Attribute, wobei noch eine Gewichtung vorgenommen wird: Seien A und B zwei Attribute, deren Klassen einander zugeordnet wurden, N_A und N_B deren Namen, D_A und D_B deren Datentypen und G_N die Gewichtung der Namen, wobei $0 < G_N < 1$. S_N bzw. S_D seien die Ähnlichkeitsfunktionen zweier Zeichenketten bzw. Datentypen. Dann ist die Ähnlichkeit $S_T(A, B)$ der Attribute A und B definiert als:

$$S_T(A, B) = G_N \cdot S_N(N_A, N_B) + (1 - G_N) \cdot S_D(D_A, D_B) \tag{2.18}$$

Das Bewertungsmaß B_T zweier einander zugeordneter Attribute A und B ist gleich 1, wenn die Datentypen identisch sind. Falls nicht, wird die Ähnlichkeit dieser Datentypen mit einer Gewichtung $0 \leq G_D \leq 1$ verrechnet.

$$B_T(A,B) = 1 - G_D + G_D \cdot S_D(D_A, D_B) \tag{2.19}$$

Die Gewichtungen G_N und G_D sind vorher unter Berücksichtigung didaktischer Gesichtspunkte festzulegen. Der Name zweier einander zugeordneter Attribute geht also nicht in die Berechnung deren Bewertung ein.

Beispiel: Die Attribute `id:long` und `ID:int` der Klassen *Benutzer* und *Person* aus dem Beispiel in Abbildung 2.1 wurden einander zugeordnet. Die Ähnlichkeit der Namen S_N beträgt 1. Sei die Ähnlichkeit der Datentypen $S_D(int, long) = 0{,}75$ und die Gewichtung der Datentypen $G_D = \frac{1}{3}$. Das Bewertungsmaß ist $\frac{11}{12}$, also etwa 88 %.

2.3.5.6. Ähnlichkeit und Bewertung von Operationen

Die Berechnung der Ähnlichkeit S_O zweier Operationen A und B kann ebenfalls nur im Kontext einer zuvor vorgenommenen Zuordnung von Klassen berechnet werden. Es wird eine Ähnlichkeit der Namen N_A und N_B der Operationen und eine Ähnlichkeit der Rückgabe-Datentypen D_A und D_B analog zur Berechnung der Ähnlichkeit von Attributen durchgeführt. Zusätzlich wird noch die Ähnlichkeit S_P der Typen der Übergabeparameter berechnet. Haben beide Operationen keine Übergabeparameter, ist $S_P = 1$. Andernfalls wird nun die Ähnlichkeit der beiden Mengen P_A und P_B der Datentypen der jeweiligen Übergabeparameter berechnet (siehe Abschnitt 2.3.3), wobei als Ähnlichkeitsfunktion das Ähnlichkeitsmaß von Datentypen verwendet wird. Die Ähnlichkeit der Operationen wird nun aus diesen drei Ähnlichkeitsmaßen wie folgt berechnet. Seien $S_N(N_A, N_B)$, $S_D(D_A, D_A)$ und $S_P(P_A, P_B)$ die Ähnlichkeitsmaße der Namen, Rückgabe-Datentypen und der Mengen der Datentypen der Übergabeparameter und sei G_N, G_{DS} und G_{PS} die Gewichtungen der Namen, Rückgabe-Datentypen und Datentypen der Übergabeparameter. Dann ist die Ähnlichkeit $S(A, B)$ der Operationen A und B definiert als:

$$S_O(A, B) = G_N \cdot S_N(N_A, N_B) + G_{DS} \cdot S_D(D_A, D_B) + G_{PS} \cdot S_P(P_A, P_B) \tag{2.20}$$

Die Gewichtungen sind vorher unter Berücksichtigung didaktischer Gesichtspunkte festzulegen, wobei deren Summe 1 betragen muss, damit stets $0 \leq S(A, B) \leq 1$ gilt.

Analog zum Bewertungsmaß von Operationen wird bei der Berechnung des Bewertungsmaßes B_O zweier einander zugeordneter Operationen die Ähnlichkeit deren Namen nicht mehr berücksichtigt:

$$B_O(A, B) = G_Z + G_{DB} \cdot S_D(D_A, D_B) + G_{PB} \cdot S_P(P_A, P_B) \tag{2.21}$$

G_Z ist dabei die Gewichtung der vorhandenen Zuordnung, also letztlich eine untere Schranke des Bewertungsmaßes für einander zugeordnete Methoden. Auch hier muss die Summer der Gewichtungen 1 sein.

Beispiel: Die Operationen `istGesperrt():boolean` und `gespert():String` der Klassen *Benutzer* und *Person* aus dem Beispiel in Abbildung 2.1 wurden einander zugeordnet. Es wurde eine Binnenmajuskel verwendet, so dass S_N gleich der Ähnlichkeit der Textsegmente $\{ist, Gesperrt\}$ und $\{gespert\}$ ist. Der Schreibfehler wird akzeptiert, so dass sich eine Ähnlichkeit von $S_N \approx 0{,}66$ ergibt. Die Rückgabetypen sind verschieden, also ist $S_D = 0$. Die Ähnlichkeit der Übergabeparameter $S_P = 1$, da die Ähnlichkeit zweier leerer Mengen gleichen 1 ist. Seien $G_N = G_Z = 0{,}5$ und $G_{DS} = G_{PS} = G_{DB} = G_{PB} = 0{,}25$, dann ist die Ähnlichkeit der Operationen $S_O \approx 0{,}83$. Das Bewertungsmaß B_O ist 0,75.

2.3.6. Ausführliches Beispiel

Sei das Verfahren wie in Abschnitt 2.4.4 beschrieben parametrisiert. Tabelle 2.1 zeigt die Zuordnungen, die für die Klassendiagramme in Abbildung 2.1 gefunden und welche Bewertungsmaße B berechnet wurden.

Die Komponenten aus der Musterlösung, für die keine Zuordnung gefunden werden konnte, werden aus Gründen der Vollständigkeit mit aufgeführt. Das Klassendiagramm der Musterlösung wird hier in einem Maß von etwa 69 % vom Klassendiagramm der Lernerlösung überdeckt. Dies entspricht der Bewertung, die Lernende für diese Lösung der entsprechenden Aufgabe erhalten würde.

2.3.7. Zusammenfassung

In den vorigen Abschnitten wurde ein Verfahren beschrieben, mit dem ein Überdeckungsmaß zwischen Klassendiagrammen ermittelt werden kann. Dieses kann zur Feedback-Generierung und Bewertung frei gezeichneter Klassendiagramme verwendet werden. Ebenso ist es möglich, nur teilweise frei gezeichnete Klassendiagramme zu bewerten, indem die für die Lernerlösung vorgegebenen Elemente und deren Entsprechungen in der Musterlösung bei der Berechnung des Überdeckungsmaßes ignoriert werden. Bei der Verwendung des Überdeckungsmaßes für Klassendiagramme (siehe Formel (2.15) auf Seite 27) als Bewertungsmaß für entsprechende Aufgaben werden zusätzliche Komponenten (also z. B. weitere Klassen) in der Lernerlösung nicht negativ bewertet. Dies ist didaktisch durchaus beabsichtigt: Lernerlösungen, in welchen die Domäne ausführlicher modelliert wird, werden so nicht schlechter bewertet.

Für eine konkrete Berechnung eines Überdeckungsmaßes muss das Verfahren parametrisiert werden. Es müssen also beispielsweise verschiedene Gewichtungen für die Berechnung von Ähnlichkeitsmaßen bestimmt werden. Grundsätzlich kann diese Parametrisierung global, also im Gesamtsystem oder lokal, also nur im Kontext einer einzigen Aufgabe vorgenommen werden.

	Musterlösung	**Lernerlösung**	***B***
Klassen	Einrichtung	-	0 %
	Bibliothek	Bibliotek	100 %
	Benutzer	Person	100 %
	Buch	Buch	100 %
	Buchexemplar	Exemplar	100 %
Generalisierung	Bibliothek → Einrichtung	-	-
Assoziationen	Bibliothek − Benutzer	Bibliotek − Person	100 %
	Bibliothek − Buch	Bibliotek − Buch	100 %
	Buch − Benutzer	Buch − Person	100 %
	Buch − Buchexemplar	-	0 %
	Buchexemplar − Benutzer	Exemplar − Person	100 %
Multiplizitäten	1:* (Bibliothek − Benutzer)	*:1 (Bibliotek − Person)	0 %
	1:* (Bibliothek − Buch)	1:* (Bibliotek − Buch)	100 %
	1:* (Buch − Benutzer)	1:* (Buch − Person)	100 %
	1:* (Buch − Buchexemplar)	-	0 %
	1:* (Buchexemplar − Benutzer)	*:* (Exemplar − Person)	50 %
Attribute	titel:String	titel:String	100 %
	autor:String	author:String	100 %
	jahr:short	-	0 %
	name:String	-	0 %
	signatur:String	sigantur:int	50 %
	id:long	ID:int	88 %
	vorname:String	vorname:String	100 %
	nachname:String	zuname:String	100 %
	geburtsdatum:Date	-	0 %
	istDozent:boolean	istDozent:Boolean	100 %
Operationen	sperren(Benutzer):void	sperren(Person)	100 %
	anlegen(Benutzer):void	-	0 %
	leihen(Buchexemplar):void	ausleihen(Buch)	75 %
	zurückgeben(Buchexemplar):void	zurückgeben(Exemplar)	100 %
	istGesperrt():boolean	gespert():String	75 %

Tabelle 2.1.: Gefundene Zuordnungen von Elemente aus der Lernerlösung (rechts) zu Elementen aus der Musterlösung (links) sowie korrespondierende Bewertungsmaße für die Klassendiagramme aus Abbildung 2.1 bei Verwendung der Parametrisierung aus Abschnitt 2.4.4

2.4. Umsetzung in CaseTrain

Die automatische Korrektur (teilweise) frei gezeichneter UML Klassendiagramme wurde nach dem dargestellten Verfahren implementiert und in das fallbasierte Trainingssystem CaseTrain [Hörnlein et al., 2009] integriert. Dabei wurden die automatisch auswertbaren Aufgabentypen in CaseTrain, beispielsweise Multiple-Choice-, Long-Menu-, Zahlen- und Wort-Fragen um einen neuen Aufgabentyp erweitert, bei welchem die Lernenden ein Klassendiagramm zeichnen müssen. Es wird derzeit die Angabe von nur einer Musterlösung zugelassen, es findet bei der Feedback-Generierung also kein Vergleich der Lernerlösung mit mehreren Musterlösungen statt. Das Verfahren ist außerdem in Absprache des an der Fallstudie beteiligten Dozenten global parametrisiert.

2.4.1. Technische Aspekte

Die Fallablaufumgebung, der CaseTrain-Player, ist eine clientseitige, browserbasierte Adobe-Flash-Anwendung. Eine Anforderung für das freie Zeichnen ist, dass der Einarbeitungsaufwand des Zeichenwerkzeugs möglichst gering sein soll, damit sich die Studierenden auf das Modellieren konzentrieren und das Arbeitsgedächtnis nicht durch zu viele nicht benötigte Optionen überlastet wird (vergleiche u.a. [Miller, 1956]). Dazu wurde der CaseTrain-UML-Editor eingesetzt, eine im Rahmen des Blended-Learning-Projekts der Universität Würzburg entwickelte Adobe-Flash-Anwendung bzw. -Komponente [Ott, 2011].

Beim Starten eines Trainingsfalles werden dem CaseTrain-Player alle Falldaten übermittelt, so dass die Bearbeitung, also auch die Feedback-Generierung bei Aufgaben, prinzipiell nur clientseitig abläuft. Für die Korrektur von Klassendiagrammen wurde CaseTrain um eine Schnittstelle zur serverseitigen Korrektur erweitert. Dies hat vor allem pragmatische Gründe, da bei einer clientseitigen Auswertung verschiedene vorhandene Bibliotheken, z. B. Stemmer, neu in ActionScript3[5] implementiert werden müssten. Auch ist die Möglichkeit einer rein clientseitigen Verwendung eines elektronischen Wörterbuchs nur aufwändig umsetzbar. Nach Bearbeitung einer Aufgabe werden die Klassendiagramme, also Muster- und Lernerlösung, in Form einer XML-Datei (XML-Schema siehe Abschnitt A.1.1.1) an ein Servlet einer Webanwendung (Apache Tomcat) übermittelt, welche das Überdeckungsmaß berechnet, ein Feedback generiert und dies zur Anzeige für die Lernenden an den CaseTrain-Player zurück übermittelt. Das Feedback besteht dabei aus dem Zahlenwert des Überdeckungsmaßes und eines Tabellendokuments zur Erklärung dessen Berechnung. Diese Schnittstelle ist dabei nicht auf den CaseTrain-Player festgelegt, so dass die Feedback-Generierung auch von anderen Systemen genutzt werden kann.

[5] ActionScript3 ist eine objektorientierte Programmiersprache auf Basis des ECMAScript-Standards zur Implementierung von Anwendungen für den Adobe Flash Player bzw. Adobe AIR (Adobe Integrated Runtime).

2.4.2. Autorenwerkzeug

Zur Aufgabenstellung wird der Domänenbeschreibung nur eine Musterlösung hinzugefügt, welche als XML-Datei vorliegen muss. Abschnitt A.1.1.2 zeigt die XML-Datei zur Musterlösung aus Abbildung 2.1. Bei der Erstellung der Aufgabe im CaseTrain-Autorenwerkzeug müssen Domänenbeschreibung und die Musterlösung mit angegeben werden. Zur Erzeugung einer solchen XML-Datei steht den Autoren der Editor, der für die Aufgabenbearbeitung im CaseTrain-Player eingebettet ist, als eigenständige Desktop- oder Webanwendung zur Verfügung. (Auf die Fall- und Aufgabenerstellung in CaseTrain wird in Abschnitt 5.2.1 auf Seite 131 detaillierter eingegangen.)

2.4.3. Aufgabenbearbeitung

Im CaseTrain-Player wird die Aufgabe dem Lernenden wie eine gewöhnliche Frage präsentiert (siehe Abbildung 2.2).

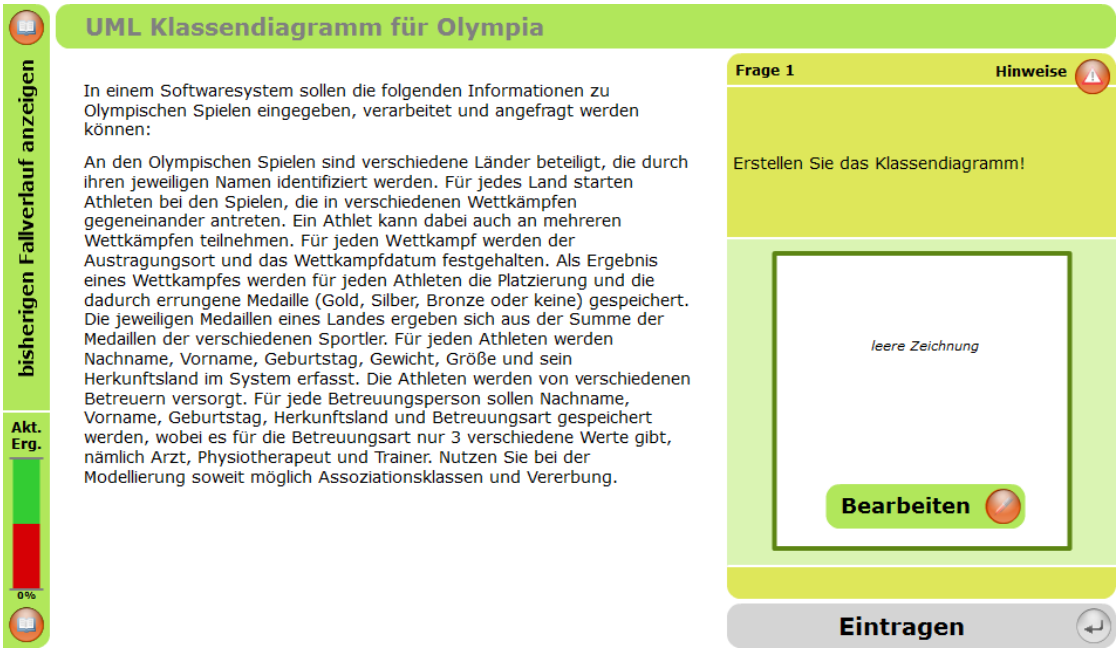


Abbildung 2.2.: Präsentation einer Aufgabe zum freien Zeichnen eines UML-Klassendiagramms im CaseTrain-Player. Die Domäne „Olympische Spiele“ soll als Klassendiagramm modelliert werden.

Die Fallablaufumgebung ist dabei grob in drei Bereiche aufgeteilt: Info-Bereich (links), Frage-Bereich (rechts oben) und Antwort-Bereich (rechts unten). Im Info-Bereich werden dem Lernenden Inhalte über den Kontext der Frage präsentiert. Dieser ist für gewöhnlich für mehrere Fragen identisch. Im Frage-Bereich steht der Fragetext. Im Antwort-Bereich befinden sich bei herkömmlichen Fragen die entsprechenden Kontrollelemente zum Beantworten einer Frage, also Checkboxen (Multiple-Choice-Fragen), Radiobuttons

(One-Choice-Fragen) oder Texteingabefelder (Wort-, Zahlen- und TextFragen). Diese Aufteilung ist dabei analog zu den Aufgabenfeldern nach [Rütter, 1982], die er als *Informationsfeld*, *Fragefeld* und *Antwortfeld* bezeichnet.

Aus technischen Gründen kann die Größe des Antwort-Bereichs im CaseTrain-Player nicht bestimmten Aufgabentypen angepasst werden. Zudem ist die Oberfläche auf eine fixe Größe von 950x550 Pixeln beschränkt, so dass der Editor zum Zeichnen der Klassendiagramme in einem zusätzlichen modalen Dialog angezeigt wird, der den maximal verfügbaren Platz einnimmt und durch das Betätigen der Schaltfläche „Bearbeiten“ geöffnet werden kann. Abbildung 2.4 zeigt den Editor zum freien Zeichnen von UML Klassendiagrammen. Im Gegensatz zu Editoren, die zum professionellen Arbeiten mit der UML eingesetzt werden, ist der CaseTrain-UML-Editor sehr einfach gehalten. Bei der Konzeptionierung wurde großer Wert auf einfache Bedienbarkeit gelegt, was sich beispielsweise auf die geringe Anzahl von Schaltflächen auswirkt.

Beim Betätigen der Schaltfläche „Speichern & Schließen“ wird das aktuelle Klassendiagramm an den CaseTrain-Server geschickt und dort gespeichert. Wenn Lernende den CaseTrain-Player nun beenden und die Bearbeitung zu einem späteren Zeitpunkt wieder aufnehmen, können sie das Diagramm an gleicher Stelle weiter bearbeiten.

Die Namen der Klassen werden als Freitext eingegeben, ebenso werden die Attribute und Operationen (beim völlig freien Zeichnen) ebenfalls durch die Eingabe von Freitext definiert. Dazu steht den Lernenden je Klasse für Attribute und Operationen jeweils ein mehrzeiliges Texteingabefeld zur Verfügung. Der Editor gibt dabei keine Hilfestellung zur syntaktisch korrekten Eingabe von Attributen und Methoden. Insbesondere gibt es keine getrennten Eingabefelder für Namen und Daten- bzw. Rückgabetypen von Attributen bzw. Operationen.

Ist das Klassendiagramm erstellt, kann es als Antwort auf die Frage eingetragen werden. Das Klassendiagramm wird dann zusammen mit der Musterlösung an das Servlet auf dem Korrekturserver geschickt, welcher das Feedback generiert und an den CaseTrain-Player zurück schickt.

2.4.4. Feedback-Generierung

Zur Feedback-Generierung wird das oben beschriene Verfahren verwendet, wobei es wie folgt global parametrisiert wurde:

- Ähnlichkeit für Wörter mit gleicher Grundform (siehe Abschnitt 2.3.2): 0,99
- maximal erlaubte Levenshtein-Distanz (siehe Abschnitt 2.3.2): 2
- maximale Ähnlichkeit synonymer Wörter (siehe Abschnitt 2.3.2): 0,99
- Bewertungsmaß für Klassen verschiedenen Typs: (siehe Abschnitt 2.3.5.1): 1 (abweichende Typen der zugeordneten Klassen werden also nicht negativ bewertet)
- Ähnlichkeit für teilweise korrekte Multiplizitäten (siehe Abschnitt 2.3.5.4): 0,5

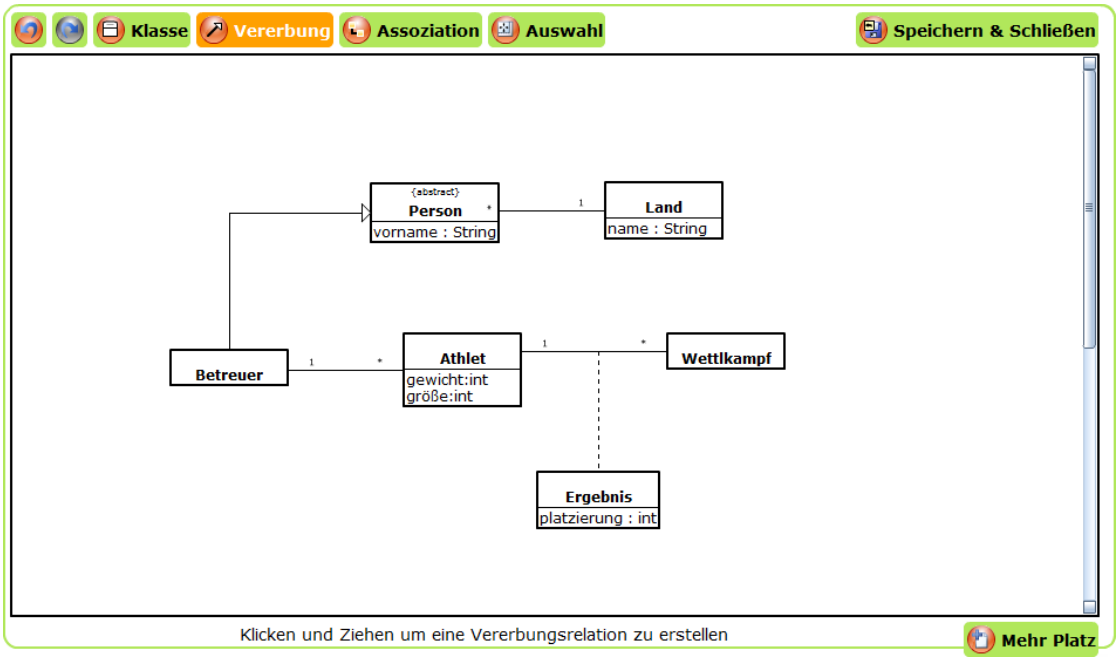


Abbildung 2.3.: Editor zum freien Zeichnen von UML Klassendiagrammen. Modelliert wird die Domäne „Olympische Spiele“.

- Ähnlichkeit von Vererbungsbeziehungen (siehe Abschnitt 2.3.5.2) mit invertierter Richtung: 0,5
- Ähnlichkeit von Datentypen (siehe Abschnitt 2.3.4) mit folgenden Kategorien:
  - numerisch
    - Gleitkommazahlen (`double`, `float`)
    - ganze Zahlen (`double`, `float`, `int`, `Number`, `byte`, `Integer`, `short`, `long`, `BigDecimal`, `BigInteger`, `AtomicInteger`, `AtomicLong`, `Zahl`)
  - Datum
    - `Date`
    - `Datum`

  Sind beide Datentypen der Kategorie „Datum“ ist die Ähnlichkeit gleich 1. Sind beide Datentypen aus der gleichen numerischen Unterkategorie (aber nicht identisch), so ist die Ähnlichkeit 0,75. Sind beide Datentypen aus verschiedenen numerischen Unterkategorien, ist die Ähnlichkeit 0,5.
- Ähnlichkeit von Attributen (siehe Abschnitt 2.3.5.5):
  - $G_N = \frac{2}{3}$ (Gewichtung der Ähnlichkeit der Namen)
- Bewertung von Attributen (siehe Abschnitt 2.3.5.5):

  - $G_D = 0{,}5$ (Gewichtung der Ähnlichkeit der Datentypen)
- Ähnlichkeit von Operationen (siehe Abschnitt 2.3.5.6):
  - $G_{NS} = 0{,}5$ (Gewichtung der Ähnlichkeit der Namen)
  - $G_{DS} = 0{,}25$ (Gewichtung der Ähnlichkeit des Rückgabeparameters)
  - $G_{PS} = 0{,}25$ (Gewichtung der Ähnlichkeit der Übergabeparameter)
- Bewertung von Operationen (siehe Abschnitt 2.3.5.6):
  - $G_{DB} = 0{,}5$ (Gewichtung der Ähnlichkeit des Rückgabeparameters)
  - $G_{PB} = 0{,}5$ (Gewichtung der Ähnlichkeit der Übergabeparameter)
- Überdeckungsmaß von Klassen (siehe Abschnitt 2.3.1), Gewichtung der Komponenten:
  - Klassen: 2
  - Vererbungsbeziehungen: 1
  - Assoziationen: 2
  - Multiplizitäten: 1
  - Attribute: 1
  - Operationen: 1

Als elektronisches Wörterbuch wurde GermaNet [Hamp und Feldweg, 1997] angebunden. Dabei handelt es sich um ein maschinenlesbares Wortnetz der deutschen Sprache, welches von der Universität Tübingen betrieben wird[6]. Damit lässt sich für jedes Wort ein sogenannte „Synset“ finden, also eine Menge an Worten, die zu diesem Wort synonym sind. Zwei Wörter A und B gelten im hier vorgestellten Verfahren genau dann als Synonym, wenn A im Synset von B oder B im Synset von A enthalten ist. Als Stemmer wird der deutsche Snowball-Stemmer von Martin Porter [Porter, 2001] eingesetzt.

2.4.4.1. Feedback-Dokument

Neben der quantitativen Bewertung ihrer Lösung (Überdeckungsmaß zur Musterlösung) und der Musterlösung selbst, erhalten die Lernenden ein Tabellen-Dokument. In diesem wird erläutert, wie diese Bewertung zustande kommt. Es werden die Komponenten der Musterlösung aufgeführt, daneben die zugeordnete Komponente der Lernerlösung, die maximal erreichbaren Punkte für die Komponente der Musterlösung, das Bewertungsmaß für diese Zuordnung und die tatsächlich erreichten Punkte für diese Komponente. Diese maximal erreichbaren Punkte ergeben sich aus der Gewichtung der Typen von Komponenten bei der Berechnung des Überdeckungsmaßes der Klassendiagramme bzw. deren Parametrisierung.

[6] www.sfs.uni-tuebingen.de/GermaNet/

	A	B	C	D	E	F
1		**Musterlösung**	**Ihre Lösung**	**max. Punkte**	**Bewertung**	**Ihre Punkte**
2	**Klassen**			**10**	**80%**	**8,0**
3	Einrichtung		-	2	0%	0
4	Bibliothek		Bibliotek	2	100%	2
5	Benutzer		Person	2	100%	2
6	Buch		Buch	2	100%	2
7	Buchexemplar		Exemplar	2	100%	2
8						
9	**Generalisierungen:**			**1**	**0%**	**0,0**
10	Bibliothek -> Einrichtung		-	1	0%	0
11						
12	**Assoziationen:**			**10**	**80%**	**8,0**
13	Bibliothek <-> Benutzer		Bibliotek <-> Person	2	100%	2
14	Bibliothek <-> Buch		Bibliotek <-> Buch	2	100%	2
15	Buch <-> Benutzer		Buch <-> Person	2	100%	2
16	Buch <-> Buchexemplar		-	2	0%	0
17	Buchexemplar <-> Benutzer		Exemplar <-> Person	2	100%	2
18						
19	**Kardinalitäten:**			**5**	**50%**	**2,5**
20	Bibliothek <-> Benutzer	1:*	*:1	1	0%	0
21	Bibliothek <-> Buch	1:*	1:*	1	100%	1
22	Buch <-> Benutzer	1:*	1:*	1	100%	1
23	Buch <-> Buchexemplar	1:*	-	1	0%	0
24	Buchexemplar <-> Benutzer	1:*	*:*	1	50%	0,5
25						
26	**Attribute:**			**10**	**64%**	**6,4**
27	Buch	titel : String	titel : String	1	100%	1,0
28	Buch	autor : String	author : String	1	100%	1,0
29	Buch	jahr : short	-	1	0%	0,0
30	Einrichtung	name : String	-	1	0%	0,0
31	Buchexemplar	signatur : String	sigantur : int	1	50%	0,5
32	Benutzer	id : long	ID : int	1	88%	0,9
33	Benutzer	vorname : String	vorname : String	1	100%	1,0
34	Benutzer	nachname : String	zuname : String	1	100%	1,0
35	Benutzer	geburtsdatum : Date	-	1	0%	0,0
36	Benutzer	istDozent : boolean	istDozent : Boolean	1	100%	1,0
37						
38	**Methoden:**			**5**	**70%**	**3,5**
39	Bibliothek	sperren(Benutzer)	sperren(Person)	1	100%	1,0
40	Bibliothek	anlegen(Benutzer)	-	1	0%	0,0
41	Benutzer	leihen(Buchexemplar)	ausleihen(Buch)	1	75%	0,8
42	Benutzer	zurückgeben(Buchexemplar)	zurückgeben(Exemplar)	1	100%	1,0
43	Benutzer	istGesperrt() : boolean	gespert() : String	1	75%	0,8
44						
45						
46	**Gesamt:**			**41**	**69%**	**28,4**

Abbildung 2.4.: Beispiel eines Feedback-Dokuments für Muster und Lernerlösung aus Abbildung 2.1

Beim Überfahren des Bewertungsmaßes erhalten Lernende eine knappe Erklärung, wie diese Bewertung zustande kommt. Beispielsweise bei der Zuordnung der Operationen `istGesperrt():boolean` und `gesperrt():String` lautet die Erklärung: „Es konnte eine entsprechende Operation gefunden werden. Der Rückgabetyp ist nicht korrekt (25 % Abzug).“

2.5. Geführtes Tutorsystem mit zunehmender Selbstständigkeit

Bei der Korrektur gibt es zwei Hauptfehlerklassen (mit entsprechenden Untertypen), die häufig kombiniert auftreten: zum einen Fehler, bei denen syntaktisch falsche oder unvollständige Klassendiagramme entstehen, und zum anderen Fehler, die auf einer falschen oder suboptimalen Konzeptionierung der Aufgabenstellung beruhen. Didaktisch ist es vorteilhaft, wenn das Vorgehen schrittweise erlernt werden kann und dann immer nur eine Fehlerklasse im Vordergrund steht. Daher wird oft zunächst die Syntax eingeübt, da sie leichter zu vermitteln ist, und dann die einfacheren konzeptionellen Probleme und zum Schluss die schwierigeren, wozu vor allem die Festlegung der Klassen gehört. Wenn die Klassen vernünftig gewählt sind, ergeben sich die restlichen Elemente (Vererbungsbeziehungen und Assoziationen, deren Eigenschaften wie Kardinalitäten und Assoziationsklassen, Auswahl und Zuordnung von Attributen und Operationen zu Klassen) oft in einer relativ systematischen und daher leichter erlernbaren Weise.

In CaseTrain wurde daher ein Instruktionsmodell umgesetzt, das den Lernenden zunächst eine starke Führung gibt und sie dann schrittweise in die Selbstständigkeit entlässt [Ifland et al., 2012]. Es orientiert sich an der Grundidee des *Cognitive Apprenticeship Modells* (CAM) [Collins et al., 1989].

Das Modell besteht aus sechs „Lehrschritten“ [Niegemann et al., 2008], die sich wiederum grob in drei Gruppen zusammenfassen lassen. Die ersten drei Schritte (Modeling, Coaching, Scaffolding) helfen den Lernenden durch Beobachtung und geführtes Üben, sich entsprechende Fähigkeiten anzueignen. In den nächsten beiden Lehrschritten (Articulation, Reflection) sollen sich die Lernenden ihrer eigenen Problemlösungsstrategien bewusst werden und diese mit den Strategien von Experten vergleichen. Im letzten Schritt (Exploration) sollen sie ermutigt werden, sich selbstständig mit den entsprechenden Inhalten zu beschäftigen.

- **Modeling**: In diesem ersten Lernschritt nehmen Lernende eine nur beobachtende Rollen ein. Sie sehen einem Experten dabei zu, wie ein Problem gelöst bzw. auf welche Weise vorgegangen wird. Sie sollen so ein konzeptuelles Modell entwickeln, um die entsprechenden Handlungen später selbst ausführen zu können.
- **Coaching**: Hier führen Lernende die Handlungen selbst aus und werden während dessen von einem Experten z. B. mittels Hinweisen und Feedback unterstützt. Das Ziel besteht darin, die Handlungen der Lernenden den Handlungen von Experten ähnlicher zu machen. Gelegentlich können einzelne Schritte auch vom Experten übernommen werden.
- **Scaffolding**: In diesem Schritt wird den Lernenden soviel Selbstständigkeit wie möglich zugestanden. Der Experte zieht seine Unterstützung möglichst weit zurück. Dieser Vorgang wird als auch *Fading* bezeichnet.
- **Articulation**: Anschließend sollen die Lernenden ermutigt werden, über ihr Wissen, Schlussfolgern und Problemlösen zu sprechen. Dies kann durch das Stellen konkreter Fragen geschehen, ebenso können die Lernenden aufgefordert werden, während der Handlung über die momentan ausgeführten Schritte zu sprechen.

- **Reflection**: In diesem Schritt sollen die Lernenden ihre Handlungen mit denen von Experten oder anderen Lernende vergleichen und entsprechend bewerten.
- **Exploration**: Zuletzt sollen die Lernenden angeleitet werden, sich selbst mit den gelehrten Inhalten zu befassen und diese selbst zu untersuchen. Dies schließt nicht nur das selbstständige Lösen von Problemen ein, sondern beispielweise auch das erstellen neuer Problemstellungen, die von anderen Lernenden gelöst werden können.

Das CAM wurde nun wie folgt umgesetzt: Nach einer allgemeinen Einführung wird Schritt für Schritt die Entwicklung eines Klassendiagramms aus einer Aufgabenstellung vorgeführt. Danach sollen die Lernenden diese Entwicklung nachahmen und auf andere Aufgabenstellungen transferieren. Das schrittweise Vorgehen wird zunächst durch ein geführtes Tutorprogramm vermittelt. Beim anschließenden freien Zeichnen wird zunächst der schwierigste Schritt, die Auswahl der Klassen, vorgegeben, so dass zuerst die restlichen, besser systematisierbaren Teilaufgaben der Klassendiagrammerstellung eingeübt werden. Erst danach werden in Übungsaufgaben bezüglich des Vorgehensmodells überhaupt keine Vorgaben mehr gemacht. Dieses Instruktionsmodell wurde im Rahmen einer Vorlesung wie folgt umgesetzt:

- **Stufe 1**: allgemeine Vermittlung der Grundlagen
- **Stufe 2**: Vorführung, wie ein Klassendiagramm aus einer Aufgabenbeschreibung erstellt wird (*Modelling*)
- **Stufe 3**: selbstständiges Nachahmen des Beispiels aus Stufe 2 und selbstständiges Erstellen von Klassendiagrammen zu neuen Beispielen in einem geführten Dialog mit automatischer Korrektur (Trainingsfall mit Multiple-Choice-Fragen) (*Coaching*)
- **Stufe 4**: teilweise freies Zeichnen eines Klassendiagramms mit automatischer Korrektur (*Scaffolding*).
- **Stufe 5**: freies Zeichnen eines Klassendiagramms (*Scaffolding*)
- **Stufe 6**: Diskussion verschiedener Modellierungsalternativen für eine Aufgabenstellung mit ihren Vor- und Nachteilen (*Articulation*, *Reflection*)

Die Vermittlung der allgemeinen Grundlagen (Stufe 1) wurde im Frontalunterricht mit Folienscript und Literaturhinweisen (nach dem Lehrbuch von Heide Balzert [Balzert, 2005]) durchgeführt. Stufe 2 wurde umgesetzt, indem der Dozent die Erstellung eines Klassendiagramms Schritt für Schritt präsentierte. Für die geführte, selbstständige Umsetzung in Stufe 3 wurde zunächst das gleiche Beispiel als Trainingsfall in CaseTrain mit Multiple-Choice-Fragen zum freiwilligen Üben durch Nachahmen bereitgestellt, um den Transfer von passivem in aktives Wissen zu unterstützen. Anschließend wurden drei weitere verpflichtende Trainingsfälle mit anderen Aufgabenstellungen gestellt, zu denen die Studierenden ein automatisch generiertes detailliertes Feedback bekamen. In Stufe 4 mussten die Studierenden in einer weiteren Übungsaufgabe ein neues Fallbeispiel in einem einfachen UML-Editor teilfrei zeichnen: Dabei war die Auswahl der Klassen

vorgegeben, so dass zunächst die restlichen, besser systematisierbaren Teilaufgaben der Klassendiagrammerstellung eingeübt wurden. Auch waren Listen von Attributen und Operationen vorgegeben, welche den Klassen zugeordnet werden konnten. Auch dazu wurde das Feedback automatisch generiert, wobei die in Abschnitt 2.4 vorgestellte Implementierung in CaseTrain verwendet wurde[7]. In Stufe 5 mussten die Studierenden schließlich eine weitere Aufgabe ohne Vorgaben frei zeichnen. Das Feedback dazu bekamen die Lernenden teilweise in der Präsenzlehre, teilweise ebenfalls in CaseTrain (siehe Abschnitt 2.6). Stufe 6 wurde in der Präsenzlehre in Kleingruppen (20-30 Teilnehmer) durchgeführt und wird hier nicht näher betrachtet.

Ein weiterer Ansatz, bei der Lehre von objektorientierter Modellierung nach der Theorie des CAM vorzugehen, wurde 2002 bereits von [Tholander und Karlgren, 2001] vorgestellt. Dabei wird der Fokus auf Aufzeichnungen, wie Experten Klassendiagramme erstellen, sowie Pattern-Bibliotheken gelegt. Ein zentraler Aspekt des Modells, das schrittweise Entlassen des Lernenden in die Selbstständigkeit (*Fading*), wurde dabei bewusst außen vor gelassen.

2.6. Evaluation und Ergebnisse

Das geführte Tutorsystem wurde in den Sommersemestern 2011 bis 2014 im Übungsbetrieb zur Vorlesung Softwaretechnik zur Erstellung von Klassendiagrammen eingesetzt. Die Gruppe der Studierenden war dabei heterogen, da die Vorlesung von verschiedenen Studiengängen besucht wird: Studierende der Informatik, der Luft- und Raumfahrtinformatik, der Wirtschaftsinformatik, der Wirtschaftsmathematik, der Mensch-Computer-Systeme und andere, die meist im ersten oder zweiten Semester waren. Die Aufgaben im Rahmen des Übungsbetriebs mussten dabei von den Studierenden bearbeitet werden, um zur abschließenden Klausur zugelassen zu werden[8]. Es nahmen jeweils etwa 300 Studierende am Übungsbetrieb teil, wobei der Anteil der Studierenden, die tatsächlich Übungsaufgaben bearbeiten, erfahrungsgemäß im Laufe des Semesters sinkt. Ein Grund dafür ist beispielsweise das gänzliche Ausscheiden von Studierenden aus dem Übungsbetrieb.

Zu folgenden Domänen mussten dabei Klassendiagramme erstellt werden:

- Universitätsbibliothek
- Fußball Bundesliga
- Schachpartie
- Fahrradverleih
- Landtagswahl

[7] Bei der Berechnung des Überdeckungsmaßes der Klassendiagramme war die Gewichtung der Klassen hier gleich 0 (vergleiche Formel 2.15 auf Seite 27).

[8] Genauer: Um die Zulassung zur Klausur zu erreichen, mussten insgesamt mindestens 50 % aller Punkte der im Übungsbetrieb gestellten Aufgaben erreicht werden.

- Brettspiel Monopoly
- Olympische Spiele

Die Domänen sind dabei bezüglich Umfang und Komplexität ähnlich. Eine vollständige Domänenbeschreibung, so wie sie den Studierenden in der Aufgabenstellung vorgelegt wurde und die Musterlösung für das Klassendiagramm finden sich in Abschnitt A.2.1 auf Seite 163. Dabei wurde für die ersten fünf Domänen jeweils ein geführter Trainingsfall gestellt, die Klassendiagramme zu den Domänen „Brettspiel Monopoly“ bzw. „Olympische Spiele“ mussten teilfrei, also mit vorgegebenen Klassen und Listen von Attributen und Operationen, bzw. völlig frei erstellt werden.

Tabelle 2.2 zeigt, in welcher Form die Aufgaben in den Sommersemestern 2011 bis 2014 gestellt wurden. Der Trainingsfall „Bibliothek“ wurde den Studierenden dabei in der Präsenzlehre vorgeführt und konnte optional zum freien Üben bearbeitet werden. Die anderen Aufgaben wurden im Rahmen des Übungsbetriebs gestellt, wobei die Studierenden pro Aufgabenblock etwa eine Woche Zeit hatten, die Aufgabe abzuschließen. Die Aufgaben zur Erstellung von Klassendiagrammen waren jeweils auf ein oder zwei Aufgabenblöcke verteilt. Bei der Bearbeitung der Trainingsfälle mit geführtem Dialog erhielten die Studierenden unmittelbar nach der Beantwortung der Multiple-Choice-Fragen eine Bewertung und entsprechendes Feedback. Zu den Aufgaben mit (teil-)freiem Dialog erhielten sie Bewertung und Feedback nach Ablauf der Bearbeitungsfrist per E-Mail.

Die Aufgabe „Monopoly“ war im Sommersemester 2011 noch nicht in einen Trainingsfall integriert. Die Studierenden mussten das Klassendiagramm hier mit dem als Desktop-Anwendung bereitgestellten Editor erstellen und mittels einer einfachen Webanwendung auf einen Korrekturserver hochladen. Die Ergebnisse wurden den Studierenden nach Ablauf der Bearbeitungsfrist per E-Mail zugeschickt.

	2011	**2012**	**2013**	**2014**
Bibliothek	o - v - g			
Fußball	ü - v - g			
Schach	ü - v - g			
Fahrrad	ü - v - g			
Landtagswahl		ü - v - g		
Monopoly	ü - u - t	ü - v - t		
Olympia			ü - v - f	

Tabelle 2.2.: Art und Weise des Einsatzes der Aufgaben in den Sommersemestern 2011 bis 2014. (Rahmen - Medium - Dialogtyp; Rahmen: ü = Übungsbetrieb, o = optional; Medium: v = virtueller Trainingsfall, u = Upload; Dialogtyp: g = geführt, t = teilfrei, f = frei)

2.6.1. Bearbeitungssstatistiken

Tabelle 2.3 zeigt aggregierte Bearbeitungssstatistiken der Sommersemester 2011 bis 2014. Je Semester konnte eine Aufgabe von einer Person höchstens einmal bearbeitet wer-

den. Bei den geführten Trainingsfällen wurden nur die Fragen berücksichtigt, die zur Erstellung des Klassendiagramms bzw. dessen Komponenten (Klassen, Vererbungsbeziehungen, Assoziationen, Kardinalitäten, Attribute, Operationen) beitragen. So wurde beispielsweise eine Frage nach Use-Cases für die Berechnung der Bewertung ignoriert. Die Bearbeitungsdauer bezieht sich dabei auf die Zeit, in welcher die Lernenden den Dialog im Browser geöffnet hatten. Wenn sie die Bearbeitung pausiert und später wieder aufgenommen hatten, wurde die Dauer dieser Pause nicht zur Bearbeitungszeit addiert.

	BIB	**FUS**	**SCH**	**FAH**	**LAN**	**MON**	**OLY**
Anzahl Vorlesungen	4	4	4	1	2	4	2
Anzahl Bearbeitungen	644	1108	1110	250	598	1035	506
Ø Dauer (h:mm:ss)	30:18	29:10	23:07	24:37	26:07	1:00:15[9]	52:51
Ø Bewertung	64 %	76 %	75 %	72 %	74 %	70 %	55 %

Tabelle 2.3.: Bearbeitungssstatistiken der Aufgaben in den Vorlesungen 2011 bis 2014 (BIB = Bibliothek, FUS = Fußball, SCH = Schach, FAH = Fahrrad, LAN = Landtagswahl, MON = Monopoly, OLY = Olympia)

Die Bearbeitungsstatistiken belegen, dass der CaseTrain-Player inklusive des Editors zur Erstellung von Klassendiagrammen von den Studierenden effektiv verwendet werden konnte. Es gab also keine grundsätzlichen Probleme bei der Erstellung der Diagramme. Weiterhin wurde festgestellt, dass die Bearbeitung einer Aufgabe mittels des (teil-)freien Dialogs mit etwa einer Stunde Bearbeitungsdauer etwa doppelt so lang dauerte wie die Bearbeitung einer Aufgabe mittels des geführten Dialogs. Dies ist aufgrund der zusätzlich benötigten Arbeitsschritte nicht überraschend. Eine weitere Ursache könnte natürlich auch sein, dass Studierende Probleme mit dem Editor hatten und daher länger brauchten, ihr internes Modell der Domäne tatsächlich abzubilden.

Jede in den geführten Trainingsfällen gestellte Frage lässt sich genau einer der oben genannten Komponenten eines Klassendiagramms zuordnen, so dass für jede dieser Komponenten eine separate Bewertung berechnet werden konnte. Tabelle 2.4 zeigt die durchschnittlichen Bewertungen der Fragen zu diesen Komponenten bzw. die entsprechende Teilbewertung aus dem in Abschnitt 2.3 beschriebenen Verfahren zur Bewertung von (teil-)frei gezeichneten Klassendiagrammen.

Durch Auswertungen dieser Art können Dozenten erkennen, ob Studierende bei bestimmten Komponenten besondere Probleme haben. Beispielsweise ist auffällig, dass die Bewertung für Assoziationen bei der Aufgabe „Schach“ deutlich niedriger als die Teilbewertung der anderen Komponenten der gleichen Aufgabe. Auch ist sie niedriger als die Teilbewertung der Assoziationen bei anderen Aufgaben, die ebenfalls im geführten Dialog gestellt wurden. Ursache dafür kann die Domäne selbst sein, die einen für die Lernenden schwierig korrekt zu modellierenden Sachverhalt beinhaltet, aber beispielsweise auch eine ungenaue Aufgabenstellung.

[9] Im Sommersemester 2011 wurde diese Aufgabe in Form eines Datei-Uploads gestellt, so dass keine Statistiken zu Bearbeitungsdauern vorliegen. Dieser Mittelwert bezieht sich also auf die Sommersemester 2012 bis 2014.

	BIB	FUS	SCH	FAH	LAN	MON	OLY
Klassen	65 %	82 %	77 %	79 %	66 %	-	76 %
Vererbungen	54 %	88 %	83 %	-	89 %	84 %	63 %
Assoziationen	63 %	73 %	56 %	75 %	85 %	79 %	61 %
Kardinalitäten	75 %	73 %	73 %	80 %	83 %	71 %	55 %
Attribute	65 %	76 %	83 %	65 %	82 %	72 %	35 %
Operationen	46 %	71 %	82 %	59 %	67 %	50 %	-

Tabelle 2.4.: Durchschnittliche Bewertung der Aufgaben nach Komponenten. Ein fehlender Eintrag bedeutet, dass entsprechenden Komponenten in der Musterlösung nicht vorhanden waren.

2.6.2. Evaluation des Verfahrens zur Feedback-Generierung

Das in Abschnitt 2.3 beschriebenen Verfahren zur Bewertung frei gezeichneter Klassendiagramme basiert auf der Zuordnung von Elementen der Lernerlösung zu Elementen der Musterlösung. Die Zuordnung der stark strukturierten Komponenten, also der Vererbungsbeziehungen, Assoziationen und Kardinalitäten ist dabei trivial und bedarf keiner gesonderten Evaluation. Die Zuordnung der anderen Komponenten, also Klassen, Attribute und Operationen dagegen ist komplexer, da zu deren Erstellung natürliche Sprache verwendet wird.

2.6.2.1. Laufzeit

Die durchschnittliche Laufzeit einer Feedback-Generierung zu einer Lernerlösung liegt im Bereich weniger Millisekunden (ms). Einige Testreihen, bei welchen für die Lernerlösungen der Aufgabe „Olympia“ aus den Semester 2013 und 2014 wiederholt Feedback generiert wurde, ergaben eine durchschnittliche Laufzeit von etwa 5 ms und eine maximale Laufzeit von 120 ms pro Lernerlösung. Dies schließt das Einlesen der Lernerlösungen von der Festplatte bzw. das Schreiben des Feedback-Dokuments auf Festplatte nicht mit ein. Für diese Vorgänge wurden durchschnittliche und maximale Laufzeiten von 8 und 70 ms (Lesen) bzw. 12 und 240 ms (Schreiben) gemessen. Diese Evaluation fand auf dem Rechner des Entwicklers[10] statt. Die gemessenen Zeiten können im Praxiseinsatz je nach eingesetzter Hardware höher sein, werden jedoch innerhalb der gleichen Größenordnung liegen. Die Dauer zur Generierung des Feedbacks ist in jedem Falle kurz genug, um den Studierenden praktisch unmittelbar Feedback geben zu können.

2.6.2.2. Evaluation der Zuordnung von Klassen

Der beschriebene Ansatz verwendet für die Zuordnung von Klassen, Attributen und Operationen die folgenden Konzepte: Grundformreduktion, Berechnung der Levenshtein-

[10] Intel Core i7@3.4 GHz Prozessor, 32 GB Arbeitsspeicher, SATA 3 SSD, Windows 7 64-bit

Distanz, Synonymität von Wörtern, Berechnung der Ähnlichkeit der durch die Klassennamen definierten Textsegmente und die Berechnung der Ähnlichkeit der enthaltenen Attribute und Operationen. Er wird im Folgenden als „komplexer Ansatz“ bezeichnet. Im Vergleich dazu (sogenannte *baseline*) wird ein Ansatz verwendet, der keines der eben genannten Konzepte verwendet, sondern lediglich genau dann zwei Klassen aus Muster- und Lernerlösung einander zuordnet, wenn deren Namen (bis auf Groß- und Kleinschreibung) identisch sind. Ansonsten verhält sich dieser Ansatz bei den Zuordnungen analog und wird im Folgenden als „trivialer Ansatz“ bezeichnet. Der triviale und der komplexe Ansatz wurde am Beispiel der Aufgabe „Olympia“, die in den Sommersemestern 2013 und 2014 gestellt wurde, vergleichend untersucht.

Eine erste Untersuchung zeigt, dass es grundsätzlich sinnvoll ist, den trivialen Ansatz bei der Zuordnung von Klassen zu erweitern: der überwiegende Teil der Lernerlösungen beinhaltet mindestens eine Klasse, deren Name zu keinem Namen einer Klasse der Musterlösung identisch ist. Dies trifft im Experiment auf insgesamt 421 der 506 Lernerlösungen (also 83,2 %) zu. Für diese Klassen kann mit dem einfachen Verfahren also keine Zuordnung gefunden werden.

Außerdem wurde untersucht, ob die vorgenommenen Zuordnungen von Klassen korrekt sind oder nicht. Eine Zuordnung zweier Klassen gilt dann als korrekt, wenn die Klassen die jeweils gleiche in der Domäne vorkommende Entität modellieren. Bei der Feedback-Generierung zu jeder Lernerlösung wird einer Klasse der Musterlösung entweder eine Klasse der Lernerlösung zugeordnet oder es wird keine Klasse der Lernerlösung zugeordnet. Diese (Nicht-)Zuordnung ist entweder korrekt oder nicht korrekt, so dass es je Lernerlösung und Klasse der Musterlösung vier mögliche Zuordnungsergebnisse gibt:

- RP (Richtig Positiv): Eine korrekte Zuordnung fand statt.
- RN (Richtig Negativ): Es fand korrekterweise keine Zuordnung statt.
- FP (Falsch Positiv): Eine unkorrekte Zuordnung fand statt.
- FN (Falsch Negativ): Es fand fälschlicherweise keine Zuordnung statt.

Die jeweiligen Zuordnungsergebnisse ergeben Mengen, so dass sie analog zu einem binären Klassifikator evaluiert werden können, wobei folgende Kennzahlen berechnet werden: Der positive Vorhersagewert (*Precision*) gibt an, wie viele der vorgenommenen Zuordnungen tatsächlich korrekt vorgenommen wurden. Die Trefferquote (*Recall*) gibt an, wie viele der korrekt vorzunehmenden Zuordnungen auch tatsächlich vorgenommen worden sind. Das F_1-Maß ist ein kombiniertes Maß, welches man durch die Bildung des harmonischen Mittels von Precision und Recall erhält. Die Falschklassifikations- oder Fehlerrate ist der Anteil der fehlerhaften Zuordnungen an allen Zuordnungsergebnissen. Diese Kennzahlen werden wie folgt berechnet:

$$Precision = \frac{|RP|}{|RP| + |FP|} \tag{2.22}$$

$$Recall = \frac{|RP|}{|RP| + |FN|} \tag{2.23}$$

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.24}$$

$$Fehlerrate = \frac{|FP| + |FN|}{|RP| + |FP| + |RN| + |FN|} \tag{2.25}$$

Die maximale Anzahl möglicher Zuordnungsergebnisse ergibt sich aus der Anzahl der Klassen der Musterlösung und der Anzahl der Lernerlösungen. Die Musterlösung im Sommersemester 2013 bestand aus 11 Klassen bei 259 Lernerlösungen, die leicht modifizierte Musterlösung im Sommersemester 2014 aus 8 Klassen bei 247 Lernerlösungen, so dass sich insgesamt eine maximale Anzahl von 4825 möglichen Zuordnungsergebnissen ergibt.

Bei Verwendung des trivialen Ansatzes wurden 2981 Zuordnungen vorgenommen, die alle als korrekt gelten (richtig positive), da die Namen der einander zugeordneten Klassen stets identisch sind und somit die gleiche Entität der Domäne modelliert wird. Fehlerhafte Zuordnungen sind bei diesem Ansatz also nicht möglich, die Anzahl der falsch positiven Zuordnungen ist damit 0. Eine manuelle Evaluation ergab allerdings, dass in 569 Fällen einer Klasse der Musterlösung fälschlicherweise keine Klasse der jeweiligen Lernerlösung zugeordnet wurde. In diesen Fällen also waren die Namen der Klassen verschieden, doch bezogen sich die Klassen auf die gleiche Entität der Domäne. Diese Ergebnisse entsprechen einem perfekten positiven Vorhersagewert von 100 %, einer Trefferquote von 84,0 %, einem F_1-Maß von 91,3 % und einer Fehlerrate von 11,8 %.

Bei Verwendung des komplexen Ansatzes gab es dagegen 3509 richtig positive, 40 falsch positive und 41 falsch negative Zuordnungen. Dies entspricht einem positiven Vorhersagewert von 98,9 %, welcher gegenüber dem perfekten positiven Vorhersagewert des trivialen Ansatzes also nur leicht sank. Die Trefferquote dagegen stieg deutlich (um 14,9 Prozentpunkte) auf 98,8 % an, so dass das F_1-Maß um 7,6 Prozentpunkte auf 98,9 % stieg. Die Fehlerrate konnte um 10,1 Prozentpunkte auf 1,7 % gesenkt werden. Tabelle 2.5 zeigt diese aggregierten Ergebnisse sowie die entsprechenden Ergebnisse je Semester.

Die Auflistung in Abschnitt A.2.2 auf Seite 171 kann einen Eindruck vermitteln, welche korrekten und unkorrekten Zuordnungen bei der Verwendung des komplexen Ansatzes vorgenommen wurden.

	Anz.	RP	FP	FN	Precision	Recall	F_1-Maß	Fehlerr.
SoSe13 trivial	2849	1699	0	291	100 %	85,4 %	92,6 %	10,2 %
SoSe13 komplex	2849	1969	16	21	99,2 %	98,9 %	99,1 %	1,3 %
SoSe14 trivial	1976	1282	0	278	100 %	82,2 %	90,2 %	14,1 %
SoSe14 komplex	1976	1540	24	20	98,5 %	98,7 %	98,6 %	2,2 %
gesamt trivial	4825	2981	0	569	100 %	84,0 %	**91,3 %**	**11,8 %**
gesamt komplex	4825	3509	40	41	98,9 %	98,8 %	**98,9 %**	**1,7 %**

Tabelle 2.5.: Ergebnisse der manuellen Evaluation der Zuordnungen von Klassen aus den Lernerlösungen zu Klassen aus der Musterlösung. Mit dem vorgestellten „komplexen“ Ansatz konnte das F_1-Maß gegenüber einem trivialen Ansatz insgesamt um etwa 7,6 Prozentpunkte auf etwa 98,9 % erhöht und die Fehlerrate (Fehlerr.) um etwa 10,1 Prozentpunkte auf etwa 1,7 % gesenkt werden. (RP = Richtig Positiv, FP = Falsch Positiv, FN = Falsch Negativ).

Dort werden die verschiedenen[11]

- 67 korrekten (richtig positiven),
- 10 unkorrekten (falsch positiven) und
- 14 fälschlicherweise nicht vorgenommenen (falsch negativen)

Zuordnungen von Klassen der Aufgabe „Olympia“ beim Experiment im Sommersemester 2013 aufgelistet. Auf eine Darstellung der 247 verschiedenen, korrekt nicht vorgenommenen (richtig negativen) Zuordnungen und der Zuordnungen aus dem Sommersemester 2014 wurde aus Platzgründen verzichtet.

Exemplarisch sei hier die häufig aufgetretene falsch positive Zuordnung der (fälschlicherweise erstellten) Klasse der Lernerlösung „Gold“ zur Klasse der Musterlösung „Ergebnis“ zu nennen. Die Klasse „Gold“ der Lernerlösung wurde dabei fälschlicherweise als Unterklasse der Klasse „Medaille“ erstellt. Für die Wörter „Gold“ und „Ergebnis“ wird eine von 0 verschiedene Ähnlichkeit wie folgt berechnet. Eines der Synonyme zu „Ergebnis“ ist laut des elektronischen Wörterbuchs „Folge“. Die Grundformreduktion ergibt „Folg“. Bei der Berechnung der Levenshtein-Distanz wird Berechnung Groß- und Kleinschreibung nicht berücksichtigt, so dass sich für „folg“ und „gold“ eine Distanz von 2 lediglich ergibt, welche gleich des Schwellwerts von 2 ist. Damit liegt eine Ähnlichkeit von 0,5 vor, so dass eine Zuordnung vorgenommen wird.
Das Beispiel zeigt, wie bei einer toleranten Vorgehensweise, die zu mehr richtig positiven Zuordnungen führen soll, auch falsch positive Zuordnungen vorgenommen werden, die sich nur schwer, oder auf Kosten von richtig positiven Zuordnungen verhindern lassen. Damit dieses beiden Klassen korrekt nicht einander zugewiesen werden, müsste entweder auf die Verwendung eines Synonymlexikons verzichtet, oder der Schwellwert für die

[11] Zwei Klassen gelten in diesem Kontext als verschieden, wenn deren Namen nicht identisch sind.

Levenshtein-Distanz müsste verringert werden. Dies würde aber wiederum zu mehr falsch negativen Zuordnungen führen, da das Verfahren weniger tolerant wäre.

Es wurde außerdem untersucht, welchen Effekt das komplexe Verfahren auf die Feedback-Generierung zu jeder Lernerlösung hatte, wobei es vier Kategorien gibt. Es gibt Lernerlösungen, für die bei Verwendung des komplexen Verfahrens zusätzlich

- ausschließlich richtige Zuordnungen (55,7 %),
- ausschließlich falsche Zuordnungen (7,5 %),
- sowohl richtige als auch falsche Zuordnungen (5,9 %) oder
- keine Zuordnungen (30,8 %)

vorgenommen wurden. Es wurde also für mehr als die Hälfte der Lernerlösungen ein ausschließlich positiver Effekt, hingegen für nur 7,6 % Lernerlösungen ein ausschließlich negativer Effekt erzielt.

Eine andere Sicht auf die Auswirkungen des komplexen Verfahrens auf der Ebene der Lernerlösungen gibt folgende Untersuchung: Je Lernerlösung wird bei der Zuordnung der Klassen ein Zuordnungsproblem gelöst. Ein solches Zuordnungsproblem gilt im Kontext dieser Evaluation als korrekt gelöst, wenn es keine falsch positiven und keine falsch negativen Zuordnungsergebnisse enthält. Bei der Verwendung des trivialen Verfahrens wurden 224 der 506 Zuordnungsprobleme (also 44,3 %) korrekt gelöst, bei der Verwendung des komplexen Verfahrens waren es dagegen 433 (also 85,6 %), was eine erhebliche Steigerung darstellt. Tabelle 2.6 zeigt die aggregierten Ergebnisse dieser Untersuchung, sowie die entsprechenden Ergebnisse je Semester.

	Anzahl ZOP	**korrekt gelöste ZOP**	**unkorrekt gelöste ZOP**	**Anteil korrekt gelöster ZOP**
SoSe13 trivial	259	126	133	48,6 %
SoSe13 komplex	259	226	33	87,3 %
SoSe14 trivial	247	98	149	39,7 %
SoSe14 komplex	247	207	40	83,8 %
gesamt trivial	506	224	282	**44,3 %**
gesamt komplex	506	433	73	**85,6 %**

Tabelle 2.6.: Ergebnisse der Untersuchung der Korrektheit der Lösungen der Zuordnungsprobleme (ZOP) von Klassen. Mit dem „komplexen“ Ansatz konnte der Anteil korrekt gelöster Zuordnungsprobleme gegenüber dem trivialen Ansatz von 44,3 % auf 85,6 % fast verdoppelt werden.

2.6.2.3. Evaluation der Zuordnung von Attributen

Analog zur Evaluation der Zuordnungen von Klassen kann auch eine Evaluation der Zuordnungen von Attributen vorgenommen werden. Eine Zuordnung zweier Attribute (innerhalb zweier einander zugeordneter Klassen) gilt dann als korrekt, wenn diese

die gleiche Eigenschaft einer Entität modellieren. Es können die gleichen Zuordnungsergebnisse wie bei den Klassen verwendet werden. Bei der manuellen Überprüfung der Zuordnungen der Attribute wurde dabei nicht auf syntaktische Korrektheit bei der Definition der Attribute geachtet, sofern nach menschlichem Ermessen klar erschien, welche Eigenschaft modelliert werden sollte. Wenn dem Attribut „gewicht : int“ aus der Musterlösung das syntaktisch unkorrekt definierte Attribut „int gewicht“ nicht zugeordnet wurde, wurde dies als falsch negatives Zuordnungsergebnis markiert.

Auch für diese Evaluation wird als Vergleich ein trivialer Ansatz gewählt, bei dem Attribute nur dann einander zugeordnet werden, wenn deren Namen (bis auf Groß- und Kleinschreibung) identisch sind. Dass auch hier die Erweiterung dieses trivialen Ansatzes nötig ist, zeigt folgende Tatsache: Bei 2437 Klassenzuordnungen enthalten beide Klassen Attribute. In 1915 (also 78,6 %) dieser Fälle enthält die Klasse der Lernerlösung mindestens ein Attribut, deren Name zum Namen keines Attributs der Musterlösung identisch ist.

Die maximale Anzahl der zu evaluierenden Zuordnungsergebnisse ergibt sich aus der summierten Anzahl der Attribute all der Klassen der Musterlösung, denen eine Klasse aus der Lernerlösung zugeordnet wurde. Für das Sommersemester 2013 ergaben sich so 2612, für im Sommersemester 2014 3745, insgesamt also 6357 mögliche Zuordnungsergebnissen.

Bei Verwendung des trivialen Ansatzes wurden 3350 richtig positive Zuordnungen vorgenommen. Die manuelle Evaluation ergab zudem 1364 falsch negative Zuordnungen. Auch hier ist die Anzahl der falsch positiven Zuordnungen gleich 0. Diese Ergebnisse entsprechen einem positiven Vorhersagewert von 100 %, einer Trefferquote von 71,1 %, einem F_1-Maß von 83,1 % und einer Fehlerrate von 21,5 %.

Bei Verwendung des komplexen Ansatzes gab es dagegen 4173 richtig positive, 70 falsch positive und 541 falsch negative Zuordnungen. Dies entspricht einem positiven Vorhersagewert von 98,4 %, welcher gegenüber dem perfekten positiven Vorhersagewert des trivialen Ansatzes also nur leicht sinkt. Die Trefferquote dagegen stieg deutlich (um 17,5 Prozentpunkte) auf 88,5 % an, so dass das F_1-Maß um 10,1 Prozentpunkte auf 93,2 % stieg. Die Fehlerrate konnte um 11,8 Prozentpunkte auf 9,6 % gesenkt werden. Tabelle 2.7 zeigt diese aggregierten Ergebnisse sowie die entsprechenden Ergebnisse je Semester.

Diese Auswertung und die zugehörigen Kennzahlen beziehen sich also auf die Korrektheit der Zuordnungen einzelner Attribute. Es lässt sich auch hier, analog zu der Evaluation der Zuordnung von Klassen (siehe Tabelle 2.6) untersuchen, welche Auswirkungen der Einsatz des komplexen Verfahrens auf die Zuordnungsprobleme von Attributen zweier zugeordneter Klassen hat. Es handelt sich also um eine grobere Betrachtungsweise, da die beobachteten Einheiten nicht mehr einzelne Attribute, sondern Mengen von Attributszuordnungen sind.

Auch hier gilt ein Zuordnungsproblem als korrekt gelöst, wenn es keine falsch positiven und keine falsch negativen Zuordnungsergebnisse enthält. Die Anzahl der zu lösenden Zuordnungsprobleme ergibt sich aus der Anzahl der vorgenommenen Zuordnungen von

	Anz.	RP	FP	FN	Precision	Recall	F_1-Maß	Fehlerr.
SoSe13 trivial	2612	1292	0	573	100,0 %	69,3 %	81,8 %	21,9 %
SoSe13 komplex	2612	1642	49	223	97,1 %	88,0 %	92,4 %	10,4 %
SoSe14 trivial	3745	2058	0	791	100,0 %	72,2 %	83,9 %	21,2 %
SoSe14 komplex	3745	2531	21	318	99,2 %	88,8 %	93,7 %	9,1 %
gesamt trivial	6357	3350	0	1364	100,0 %	71,1 %	**83,1 %**	**21,5 %**
gesamt komplex	6357	4173	70	541	98,4 %	88,5 %	**93,2 %**	**9,6 %**

Tabelle 2.7.: Ergebnisse der manuellen Evaluation der Zuordnungen von Attributen aus den Lernerlösungen zu Attributen aus der Musterlösung. Mit dem vorgestellten „komplexen“ Ansatz konnte das F_1-Maß gegenüber einem trivialen Ansatz insgesamt um etwa 10,1 Prozentpunkte auf etwa 93,2 % erhöht und die Fehlerrate (Fehlerr.) um etwa 7,3 Prozentpunkte auf etwa 5,9 % gesenkt werden. (RP = Richtig Positiv, FP = Falsch Positiv, FN = Falsch Negativ).

Klassen, bei denen sowohl die Klasse der Musterlösung als auch die Klasse der Lernerlösung mindestens ein Attribut beinhaltete. Es ergaben sich so insgesamt 2437 zu lösende Zuordnungsprobleme. Mit dem trivialen Verfahren konnten davon 1433 (also 58,8 %) korrekt gelöst werden, bei Verwendungen des komplexen Verfahrens dagegen 3344 (also 88,3 %). Tabelle 2.8 zeigt die aggregierten Ergebnisse dieser Untersuchung, sowie die entsprechenden Ergebnisse je Semester.

	Anzahl ZOP	korrekt gelöste ZOP	unkorrekt gelöste ZOP	Anteil korrekt gelöster ZOP
SoSe13 trivial	974	556	418	57,1 %
SoSe13 komplex	974	785	189	80,6 %
SoSe14 trivial	1463	877	586	59,9 %
SoSe14 komplex	1463	1207	256	82,5 %
gesamt trivial	2437	1433	1004	**58,8 %**
gesamt komplex	2437	1992	445	**81,7 %**

Tabelle 2.8.: Ergebnisse der Untersuchung der Korrektheit der Lösungen der Zuordnungsprobleme (ZOP) von Attributen. Mit dem „komplexen“ Ansatz konnte die Quote Anteil der korrek gelösten Zuordnungsprobleme gegenüber dem trivialen Ansatz von 58,8 % auf 81,7 % erhöht werden.

Auch bei den Zuordnungen der Attribute lässt sich untersuchen, welchen Effekt das komplexe Verfahren auf die Feedback-Generierung zu jeder Lernerlösung hatte. Es gibt also auch hier Lernerlösungen, für die bei Verwendung des komplexen Verfahrens zusätzlich

- ausschließlich richtige Zuordnungen (66,2 %),
- ausschließlich falsche Zuordnungen (2,1 %),
- sowohl richtige als auch falsche Zuordnungen (8,9 %) oder
- keine Zuordnungen (22,7 %)

vorgenommen wurden. Es wurde also für knapp zwei Drittel der Lernerlösungen ein ausschließlich positiver Effekt, hingegen für nur 2,1 % Lernerlösungen ein ausschließlich negativer Effekt erzielt.

Auch wurde auf der Ebene der Lernerlösungen untersucht, ob bei der Feedback-Generierung alle zur entsprechenden Lernerlösung gehörenden Zuordnungsprobleme korrekt gelöst wurden. Bei der Verwendung des einfachen Verfahrens wurden nur bei 41 von 506 (also 8,1 %) der Lernerlösungen die Zuordnungsprobleme von Attributen völlig korrekt gelöst. Bei Verwendung des komplexen Verfahrens waren es 211 (also 41,7 %).

Eine Evaluation der Zuordnung von Operationen wurde nicht durchgeführt. Da hier die gleichen Konzepte wie bei der Zuordnung von Operationen angewandt werden, sind ähnliche Ergebnisse wahrscheinlich.

2.6.2.4. Diskussion

Es wurde gezeigt, dass ein trivialer Ansatz, der nur die Gleichheit verschiedener Komponenten berücksichtigt, nicht aber eine durch verschiedene Ansätze berechnbare Ähnlichkeit, in der Praxis nicht ausreicht, da sehr viele Lernerlösungen unter anderem mit Rechtschreib- oder Tippfehlern behaftet sind, oder verschiedene Flexionen oder Synonyme der Wörter verwendet werden.
Der beschriebene komplexe Ansatz führt dagegen zu teilweise erheblichen Verbesserungen und liegt bei der Zuordnung von Klassen mit einem F_1-Maß von 98,9 % und einer Fehlerrate von 1,7 % schon nahe am Optimum. Zwar liegt auch das F_1-Maß bei der Zuordnung der Attribute bei 93,2 %, doch ist die Fehlerrate mit 9,6 % relativ hoch. Dies hat zur Folge, dass ein relativ großer Anteil der generierten Feedbacks diesbezüglich nicht völlig fehlerfrei ist.

Grundsätzlich sind falsch negative Zuordnungsergebnisse das hauptsächliche Problem. Es werden also Komponenten trotz gleicher Semantik fälschlicherweise nicht als ähnlich erkannt und somit nicht einander zugeordnet. Dies ist ein für dieses Verfahren inhärentes Problem, welches aufgrund des Variantenreichtums natürlicher Sprache sehr wahrscheinlich nicht vollständig gelöst werden kann.

Ein weiteres Problem bei diesem Ansatz stellen Varianten der Musterlösung dar. Zwar können weitere Musterlösungen sehr leicht eingebunden werden, wobei jene Musterlösung als Referenzlösung ausgewählt wird, zu der die Lernerlösung das höchste Überdeckungsmaß aufweist, doch kann das Problem damit in der Praxis nicht vollständig gelöst werden. Auf diese Problematik wird in Abschnitt 2.7.5.2 noch eingegangen.

Eine Schwäche dieser Evaluation ist die Tatsache, dass zu lediglich einer Domäne eine Aufgabe im freien Dialog gestellt und evaluiert wurde. Die Ergebnisse von Untersuchungen zu Bearbeitungen weiterer Domänen und entsprechender Musterlösungen können

also von den hier erhaltenen Ergebnissen abweichen. Die Evaluation belegt jedoch, dass mit dem verwendeten Verfahren im Praxiseinsatz bei geeigneter Domäne und Musterlösung in jedem Falle gute Ergebnisse erzielt werden können. Es ist darauf hinzuweisen, dass weder die Domänenbeschreibungen noch die entsprechenden Musterlösungen der in den Experimenten verwendeten Aufgaben auf das vorgestellte und evaluierte Verfahren hin optimiert, sondern ohne Rücksicht darauf erstellt wurden. Es ist davon auszugehen, dass auch bessere Ergebnisse erzielt werden können, wenn Domäne, Musterlösung(en) und Aufgabenstellung auf das Verfahren angepasst werden.

2.6.3. Umfrageergebnisse

Nach der Beendigung der Experimente wurden jeweils Befragungen der Studierenden durchgeführt.

2.6.3.1. Sommersemester 2013

Etwa zwei Wochen nach Bearbeitung der Aufgaben wurde den Studierenden ein schriftlicher Fragebogen vorgelegt (siehe Anhang Seite 181), in welchem auch konkrete Fragen zur Bearbeitung der Aufgaben über Klassendiagramme gestellt wurden. Dabei wurden die Studierenden sowohl zum geführten Dialog, als auch zum (teil-)freien Zeichnen befragt. Die Studierenden sollten bestimmte Aspekte der Anwendung mit Schulnoten (1-6) bewerten. Tabelle 2.9 zeigt die Ergebnisse dieser Umfrage.

	GD	**FZ**
Aspekt	**Ø Note**	**Ø Note**
Konzept des Tools, unabhängig von der Implementierung	2,15	2,25
Umsetzung des Tools, Gesamtnote	2,52	2,61
Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)	1,96	2,47
Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)	3,11	3,57
Aspekt Inhalt der Übungsaufgaben	2,14	2,18
Aspekt Fairness der Bewertung der Übungsaufgaben	2,48	2,59
Aspekt hilfreiche Erklärungen als Text oder nützliches Feedback	2,73	2,95

Tabelle 2.9.: Umfrageergebnisse zu den Aufgaben zur Erstellung von Klassendiagrammen mit geführtem Dialg (GD) und freiem Zeichnen (FZ) im Sommersemester 2013; die verschiedenen Aspekte wurden mit Schulnoten (1-6) bewertet (n=240)

2.6.3.2. Sommersemester 2014

Etwa zwei Wochen nach Bearbeitung der Aufgaben, wurden die Studierenden gebeten, einen Online-Fragebogen auszufüllen, in welchem auch konkrete Fragen zum Erstellen

von Klassendiagrammen gestellt wurden (vollständiger Fragebogen siehe Anhang Seite 182). Dabei wurde nicht zwischen geführtem und (teil-)freien Dialog unterschieden. Die Studierenden sollten bestimmte Aspekte der Anwendung mit Schulnoten (1-6) bewerten. Tabelle 4.7 zeigt die Ergebnisse dieser Umfrage. Die Frage, ob Aufgaben zur Erstellung von Klassendiagrammen mit CaseTrain in Zukunft auch weiterhin im Übungsbetrieb eingesetzt werden sollten, beantworteten 90 % der Studierenden mit „Ja“.

Aspekt	**Ø Note**
Konzept des Tools, unabhängig von der Implementierung	1,99
Umsetzung des Tools, Gesamtnote	2,66
Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)	2,74
Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)	2,72
Aspekt Inhalt der Übungsaufgaben	2,57

Tabelle 2.10.: Umfrageergebnisse zum (teil-)freien Zeichnen von Klassendiagrammen im Sommersemester 2014; die verschiedenen Aspekte wurden mit Schulnoten (1-6) bewertet (n=93)

Die Studierenden konnten in der elektronischen Umfrage mit einem Freitext angeben, was ihnen gut und was ihnen nicht gefallen hat. Die Fragen wurden von 28 (positive Aspekte) bzw. 41 Studierenden (negative Aspekte) beantwortet. Die genannten Aspekte wurden zusammengefasst und solche, die mehr als einmal genannt wurden, sind in den Tabellen 2.11 bzw. 2.12 dargestellt (vollständige Antworten siehe A.2.5.4 auf Seite 190).

Aspekt	**Häufigkeit**
gute Einführung / steigende Komplexität	4
elektronische Bearbeitung	4
Benutzerfreundlichkeit	4
Umsetzung	3
Konzept	3
Inhalt der Aufgaben	2
unmittelbares Feedback (geführter Dialog)	2
praktische Anwendung / Interaktivität	2

Tabelle 2.11.: Die am häufigsten genannten positiven Aspekte zur Erstellung von Klassendiagrammen. Einmalig genannt wurden noch: Lernerfolg und gute Fehlermeldungen (insgesamt 28 Antworten)

2.6.3.3. Diskussion

Das Konzept der Anwendung wurde von den Studierenden im Schnitt gut bewertet. Es gibt allerdings auch Studierende, die das Konzept einer automatischen Korrektur von Klassendiagrammen grundsätzlich ablehnen. Die Umsetzung der Anwendung wurde dagegen etwas schlechter mit gut bis befriedigend bewertet. Ursache dafür könnte

Aspekt	Häufigkeit
Lösungsvarianten werden nicht akzeptiert	10
ungenaue Aufgabenstellung o. fehlende Hinweise	9
Bedienung des Editors	9
automatische Korrektur im Allgemeinen	7
Organisation Übungsbetrieb	3
Ladezeiten	2

Tabelle 2.12.: Die am häufigsten genannten negativen Aspekte zur Erstellung von Klassendiagrammen. Einmalig genannt wurden noch: zu schwere Aufgaben, fehlendes Feedback, unklares Feedback und die Verwendung von Flash (insgesamt 41 Antworten)

die Tatsache sein, dass alternative Lösungsvarianten nicht akzeptiert werden, die Aufgabenstellungen teilweise zu ungenau waren oder einige Studierende Probleme mit der Bedienung des Editors hatten. Hervorzuheben sind die Ergebnisse zur Generierung des Feedbacks im Sommersemester 2013: die Fairness der Bewertungen erscheint den Studierenden beim freien Zeichnen ähnlich fair wie bei geschlossenen Multiple-Choice-Fragen. Auch erschienen ihnen Erklärungen und Feedback jeweils in befriedigendem Maße hilfreich.

2.7. Zusammenfassung und Ausblick

Es wurde ein Verfahren zur automatischen Generierung von Feedback zur Erstellung von UML Klassendiagrammen vorgestellt, welches auf einem Vergleich einer Musterlösung und der Lösung einer Menge von Zuordnungsproblemen beruht. Um abweichende Benennung verschiedener Komponenten der Lernerlösung berücksichtigen zu können, werden in diesem Verfahren einige Techniken des NLP verwendet. Dieses Verfahren wurde als Erweiterung des Trainingssystems CaseTrain implementiert und als Teil eines geführten Tutorsystems in 4 Semestern im Übungsbetrieb einer Vorlesung eingesetzt und evaluiert. Es wurde gezeigt, dass das beschriebene Verfahren gegenüber einem trivialen Ansatz zu deutlichen Verbesserungen bei der Korrektheit der Feedback-Generierung führt. Von den Studierenden wurden Konzept und praktischer Einsatz gut bis befriedigend bewertet, wobei herauszuheben ist, dass laut der Umfrage im Sommersemester 2014 90 % der Studierenden einen weiteren Einsatz des eingesetzten Trainingssystems bei der Erstellung von UML Klassendiagrammen befürworten. Dennoch ergeben sich noch einige Möglichkeiten, das Trainingssystem weiter zu verbessern.

2.7.1. Erweiterungen durch regelbasierte Prüfungen

Im vorgestellte Trainingssystem findet neben dem Vergleich mit Musterlösungen keine weitere Überprüfung der Lernerlösungen statt. Es sind jedoch eine Reihe von Überprü-

fungen durch allgemein definierte Regeln denkbar, mit welchen sich weiteres hilfreiches Feedback generieren ließe. Die Implementierung dieser Erweiterungen hat hohe Priorität, da entsprechende Eigenschaften der Lernerlösungen mit dem aktuellen Ansatz in keiner Weise berücksichtigt werden.

2.7.1.1. Anzahl überzähliger Elemente

Es ist durchaus beabsichtigt, dass sich überzählige Elemente in der Lernerlösung (Klassen, Attribute, Operationen) bei der Berechnung der Bewertung nicht negativ auswirken, da Studierende, die die Domäne detaillierter als gefordert modellieren, nicht bestraft werden sollen. Doch ist es sinnvoll, übermäßig viele Elemente negativ zu bewerten, da der Sinn bei der Modellierung auch in einer Abstraktion bzw. Konzentration auf die wichtigsten Elemente ist. So können entsprechende Fehlermeldungen gegeben werden, wenn die Anzahl der Elemente die der Musterlösungen weit überschreitet.

Dass überzählige Assoziationen und Vererbungsbeziehungen zwischen korrekt angegebenen Klassen, die also tatsächlich unkorrekt sind, nicht negativ bewertet werden, ist eine Schwäche des vorgestellten Ansatzes. Auch diese Problematik kann mit regelbasierten Verfahren behoben werden, indem entsprechende Klassen schlechter bewertet werden.

2.7.1.2. Einhaltung von Konventionen

Bei der Erstellung von UML Klassendiagrammen gibt es gewisse Konventionen, die vom vorgestellten Verfahren nicht berücksichtigt werden. Beispielsweise sollten Namen von Klassen stets Substantive im Singular sein und mit einem großen Buchstaben beginnen. Namen von Attributen und Operationen sollten dagegen mit einem kleinen Buchstaben beginnen (vergleiche [Balzert, 2005, S.24]). Hier lassen sich Regeln aufstellen und bei der Generierung des Feedbacks überprüfen, wobei den Lernenden entsprechende Rückmeldungen gegeben werden können.

Eine implizite Konvention ist die gleichmäßige räumliche Verteilung der Klassen auf der Zeichenfläche. Diese könnte ebenso überprüft werden wie Kreuzungen von Assoziationen und Vererbungsbeziehungen, deren Vorkommen die Übersichtlichkeit des Klassendiagramms einschränkt.

Eine Konvention, die bei der Verwendung natürlicher Sprache stets gilt, ist die der korrekten Rechtschreibung. Das vorgestellte Verfahren ist hier durch die Berechnung der Levenshtein-Distanz und die Grundformreduktion zu einem gewissen Grad tolerant, doch sollten Lernende zumindest auf Schreib- und Tippfehler hingewiesen werden. Das in A.2.2 ab Seite 171 dokumentierte, nicht seltene Vorkommen von teilweise ausgefallenen Schreibweisen von Begriffen („Pyhsiothreapeut“) belegt einen Bedarf entsprechender Überprüfungen.

2.7.2. Erweiterung durch Rückfluss von Informationen aus manueller Nachkorrektur

CaseTrain bietet grundsätzlich die Möglichkeit einer manuellen Nachkorrektur von Aufgaben durch Lehrpersonal. Für UML Klassendiagramme ergeben sich hier interessante Erweiterungsmöglichkeiten. Menschliche Korrektoren können falsch negative und falsch positive Zuordnungen der automatischen Auswertung einer Lernerlösung erkennen. Eine Erweiterung der Komponente zur manuellen Korrektur sollte es nun erlauben, diese Fehler als solche zu annotieren, wobei sich das System diese Annotation merkt. So kann nicht nur die aktuell bearbeitete Lernerlösung neu ausgewertet werden, sondern auch zukünftige (oder während des Übungsbetriebs bereits eingereichte) Lernerlösungen. Ein Produkt der Annotationen falsch negativer Zuordnungen wäre eine Art „Synonymlexikon“ für diese Domäne. Aus den Annotationen falsch positiver Zuordnung ließe sich eine analoge Liste automatisch erstellen. Bei Verwendung dieser Listen bei der Lösung der Zuordnungsprobleme ließen sich damit noch bessere Ergebnisse erzielen.

2.7.3. Erweiterung zur Verbesserung des Überdeckungsmaßes

Das erwähnte Problem überzähliger Assoziationen und Vererbungsbeziehungen zwischen korrekt angegebenen Klassen ließe sich auch lösen, indem deren Anzahl bei der Berechnung der Bewertungsfunktion von Klassen (siehe Abschnitt 2.3.5.1) berücksichtigt wird.

2.7.4. Erweiterungen zur Verbesserung der Zuordnungsergebnisse

Die Lösung der Zuordnungsprobleme im beschriebenen Verfahren ist, auch wenn sie im durchgeführten Experiment bei den Klassen schon nahe am Optimum ist, noch fehlerbehaftet. Um die noch auftretenden Fehler zu verringern, können folgende Erweiterungen implementiert werden.

2.7.4.1. Dynamisches Maximum der Levenshtein-Distanz

Wie in Abschnitt 2.3.2 beschrieben, gibt es einen statischen Schwellwert, welchen die Levenshtein-Distanz zweier Zeichenketten nicht überschreiten darf, damit für diese eine Ähnlichkeit größer 0 berechnet wird. Dieser Schwellwert, welcher in der Praxis bei einem Wert von 2 oder 3 liegt, ist nötig, damit zwei semantisch verschiedene Wörter (beispielsweise „Bus“ und „Buch“) nicht als ähnlich gelten. Dies hat jedoch zur Folge, dass identische Wörter, die beispielsweise lediglich mit mehreren Tippfehlern versehen sind, als nicht ähnlich gelten. Beim Experiment im Sommersemester 2013 traf dies beispielsweise auf die Klasse „Physiotherapeut“ der Musterlösung und die Klasse „Pyhsiothreapeut“ der Lernerlösung zu, deren Namen eine Levenshtein-Distanz von 4 aufweisen (weitere Beispiele siehe Abschnitt A.2.2 auf Seite 174.)

Hier wäre es nun sinnvoll, den Schwellwert der Levenshtein-Distanz dynamisch in Abhängigkeit der Wortlängen anzupassen. So könnte der zulässige Schwellwert beispielsweise

auf das gerundete Viertel der Wortlänge gesetzt werden, was im genannten Beispiel einer erlaubten Levenshtein-Distanz von 4 entspräche. Der konkrete Schwellwert sollte mit Hilfe weiterer empirischer Untersuchungen festgelegt werden. Mit dieser Erweiterung könnten viele falsch negative Zuordnungsergebnisse vermieden werden, wobei durch die Berücksichtigung der Wortlänge die Wahrscheinlichkeit, dabei auch viele falsch positive Zuordnungen zu erzeugen, eher als gering einzuschätzen ist.

2.7.4.2. Automatische Dekomposition von Komposita

Die Namen der in den Klassendiagrammen enthaltenen Elemente können zusammengesetzte Wörter, sogenannte Komposita sein. Bei diesen Wörtern, die durch sogenannte Komposition entstehen, werden mehrere Morpheme aneinandergereiht. Durch die Aneinanderreihung dieser „kleinsten bedeutungstragenden Elemente der Sprache“ ergibt sich ein neues Wort, dessen Bedeutung umfangreicher ist als die Summer der Bedeutungen seiner Bestandteile (vergleiche [Schunk, 2002, S. 113, 139]). In der deutschen Sprache sind diese, beispielsweise im Gegensatz zur englischen, sehr häufig. Für den Kontext von UML Klassendiagrammen bedeutet dies, dass für den Namen entsprechender Elemente keine Binnenmajuskeln (also CamelCase) benutzt wird, da es sich um ein Wort handelt, welches im natürlichen Gebrauch der Sprache auch unverändert benutzt wird. Dies wiederum hat zur Folge, dass das vorgestellte Verfahren das Wort momentan nicht in seine Bestandteile aufspalten kann, so dass diese nicht bei der Berechnung der Ähnlichkeit berücksichtigt werden können.

Ein Beispiel ist hier die Klasse „Betreuungsart“ aus der Musterlösung zur Domäne „Olympia“ (siehe Seite 170), welches (zuzüglich des sogenannten Fugenlauts „s“) aus den Morphemen „Betreuung“ und „Art“ zusammengesetzt ist. Das beschriebene Verfahren erkennt diese beiden Bestandteile nicht als solche und führt nur einen Vergleich auf Basis der vollständigen Zeichenkette durch. Beinhaltet eine Lernerlösung nun eine Klasse namens „Art“ oder „Betreuungsmethode“, wird fälschlicherweise keine Ähnlichkeit größer 0 berechnet, obwohl nach menschlichem Ermessen Klassen mit diesen Namen die gleiche Entität der Domäne modellieren. Gleiches gilt für die Klasse „Wettkampf“ aus der Musterlösung und Klassen namens „Wettbewerb“ aus der Lernerlösung. Bei Attributen trifft dies im durchgeführten Experiment auf das „wettkampfsdatum“ in der Klasse „Wettkampf“ zu.

Eine Lösung für dieses Problem stellt die automatische Dekomposition der zusammengesetzten Wörter dar. Dazu existieren in der Literatur bereits Ansätze (beispielsweise [Moulinier et al., 2001] und [Braschler und Ripplinger, 2003]), die entsprechend eingebunden werden müssten. Sind die Wörter zerlegt, kann anschließend eine Ähnlichkeit auf Grundlage der Bestandteile, also der entsprechenden Textsegmente (siehe Abschnitt 2.3.3), berechnet werden.

Bei Verwendung dieser Erweiterung des Verfahrens hätten bei den Experimenten in den Sommersemestern 2013 und 2014 allein für die Zuordnung zum Attribut „wettkampfsdatum“ insgesamt etwa 180 falsch negative Zuordnungen vermieden werden können. Sehr

viele Studierende gaben hier lediglich „datum“ oder „austragungsdatum“ an. Eine Vermeidung der falsch negativen Zuordnungen für allein dieses Attribut hätten den Recall (unter der Voraussetzung keiner zusätzlichen falsch positiv vorgenommenen Zuordnungen) um knapp 4 Prozentpunkte und das F_1-Maß um gut 2 Prozentpunkte erhöht und die Fehlerrate um knapp 3 Prozentpunkte gesenkt. Auch bei dieser Erweiterung ist die Wahrscheinlichkeit, viele falsch positive Zuordnungen zu erzeugen, eher gering einzuschätzen.

2.7.5. Erweiterte Musterlösungen

2.7.5.1. Explizite Synonyme

Zwar wird im vorgestellten Verfahren ein elektronisches Wörterbuch angebunden, doch beinhalten solche vor allem bei domänenspezifischen Fachsprache üblicherweise wenige Einträge bzw. entsprechende Synonymbeziehungen. Da auch die anderen Techniken, also die Berechnung der Levenshtein-Distanz und die Grundformreduktion nicht hilfreich sind, werden synonym benannte Klassen aus der Lernerlösung nicht berücksichtigt. Eine Lösung für dieses Problem sind „erweiterte Musterlösungen“. Hier können also für verschiedene Elemente, also Klassen, Attribute und Operationen, alternative Namen eingegeben werden. Eine Komponente wird also nicht mehr durch einen Namen, sondern durch mehrere definiert. Bei der Lösung des entsprechenden Zuordnungsproblems wird die Ähnlichkeit zu einem solchen Element dann durch die maximale Ähnlichkeit zu einem dieser Namen bestimmt, so dass eine Zuordnung zum synonym benannten Element möglich wird. In der Musterlösung, die im Feedback zu der entsprechenden Lernerlösung angezeigt wird, kann dann die entsprechende Namensalternative verwendet werden, wobei auch eine natürlichsprachige Erklärung generiert werden kann, die auf die verschiedenen Möglichkeiten der Benennung hinweist.

2.7.5.2. Variable Zugehörigkeit von Operationen zu Klassen

Wie bereits in Abschnitt 2.6.2.4 angedeutet, stellen Varianten der Musterlösung ein Problem dar. Werden durch die Domäne beispielsweise Operationen beschrieben, für deren Zugehörigkeit zu einer Entität (also Klasse) mehrere Möglichkeiten existieren, so ergibt sich hier eine exponentiell steigende Anzahl möglicher Musterlösungen: Beschreibt die Domäne o Operationen, welche jeweils in k verschiedenen Klassen enthalten sein können, so ergeben sich k^o verschiedene Varianten einer Musterlösung. Es ist daher nicht praktikabel, für alle diese Varianten jeweils eine Musterlösung zu erstellen.

Auch dieses Problem kann umgangen werden, indem erweiterte Musterlösungen erstellt werden. Kann eine Operation in mehreren Klassen vorkommen, wird sie in jeder dieser Klassen angelegt. Um anzuzeigen, dass es sich um eine ausschließende Disjunktion handelt, die Operation also nur in einer der Klassen vorkommen sollte und nicht in beiden, wird sie zusätzlich markiert. Alle diese markierten Operationen mit identischem Namen werden bei der Berechnung des Überdeckungsmaßes nun gesondert behandelt: Alle

Klassen der Lernerlösung, die jeweils einer solchen Klasse der erweiterten Musterlösung zugeordnet wurden, die die Operation enthalten, werden nacheinander betrachtet. Enthält eine der Klassen die Operation, wird diese aus den anderen Klassen der erweiterten Musterlösung „entfernt“ und bei der Lösung der Zuordnungsprobleme von Operationen dieser Klassen nicht länger berücksichtigt. Auch in der zum Feedback zu dieser Lernerlösung angezeigten Musterlösung wird die Operation ausschließlich in der Klasse angezeigt, in der sie auch in der Lernerlösung enthalten ist. Ist sie nicht in der Lernerlösung enthalten, kann sie in einer zuvor in der Musterlösung markierten bevorzugten Klasse angezeigt werden. Zusätzlich kann auch hier für das automatisch generierte Feedback ein natürlichsprachiger Text generiert werden, in dem erklärt wird, dass die entsprechende Operation in mehreren Klassen vorkommen kann.

3. UML-Aktivitätsdiagramme und Programmcode

Wie in Kapitel 2 beschrieben, ist die objektorientierte Modellierung mittels der UML2 ein wichtiger Bestandteil der Lehre in der Informatik-Ausbildung. Neben den Klassendiagrammen für die statische Modellierung werden in der Hochschulbildung vor allem Aktivitätsdiagramme zur dynamischen Modellierung eingesetzt. Das Erstellen von solchen Diagrammen fällt Studierenden in der Regel nicht leicht und sollte daher praktisch geübt werden, wobei auch hier entsprechende Korrekturen durch Lehrpersonal aufwändig sind.

Im Folgenden verstehen wir unter einem Aktivitätsdiagramm eine Menge von Aktivitäten. Jede Aktivität modelliert dabei einen bestimmten Vorgang (Routine) und besteht unter anderem aus Aktionen, Elementen zur Abbildung von Kontrollstrukturen und entsprechenden Verbindungslinien. Eine Aktion beschreibt durch ihren Namen ebenfalls einen Vorgang. Dieser Name kann identisch zum Namen einer im Aktivitätsdiagramm enthaltenen Aktivität sein, was einen Aufruf dieser Aktivität bzw. der durch sie modellierten Routine darstellt. Ist der Name nicht identisch zu dem einer Aktivität, wird dieser Vorgang nur durch den Namen verbal beschrieben (Beispiele ab Seite 174).

3.1. Didaktische Motivation zur Verbindung von Aktivitätsdiagrammen und Programmcode

Auch die Programmierung wird in der Informatik-Ausbildung gelehrt und eingeübt. Zwar lassen sich Aktivitätsdiagramme bei geeigneter Notation in Programmcode übersetzen, doch ist dies nicht der ursprüngliche Zweck der Erstellung der Aktivitätsdiagramme. Sie betonen im Gegensatz zu Programmcode die Verständlichkeit der entsprechenden zu modellierenden Routine und sind daher kein Abbild des Programmcodes, sondern ein verständlich geschriebenes Modell des Programms auf einem höheren Abstraktionsniveau.

Das Modell einer Routine, und somit die entsprechende Aktivität, besteht auf der denkbar höchsten Abstraktionsebene aus nur einer Aktion, welche implizit alle Subroutinen einschließt, diese jedoch nicht explizit darstellt. Diese Aktion kann nun in mehrere Subroutinen unterteilt werden, welche erneut durch einzelne Aktionen, Kontrollstrukturen und Verbindungslinien dargestellt werden. Für jede dieser Aktionen kann nun eine separate Aktivität erstellt werden, die wiederum zunächst nur aus einer Aktion besteht

und weiter aufgespalten werden kann. Führt man diesen Prozess rekursiv weiter, ergibt sich eine Hierarchie, in der auf unterster Ebene letztlich Aktivitäten stehen, die eine direkte Abbildung eines Programmcodes sind, da sie aus Aktionen bestehen, die sich nicht weiter aufspalten lassen (Abbruchbedingung der Rekursion).

Natürlich ist diese konkrete Ebene nicht das Ziel einer Modellierung durch Aktivitätsdiagramme. Doch ergibt sich daraus ein interessantes Top-Down Prozessmodell zur Programmentwicklung, welches Anfängern zeigen kann, dass Aktivitätsdiagramme letztlich äquivalent zu Programmcode sind, wenn man in dieser Abstraktionshierarchie weit genug nach unten sieht. Dieses Prozessmodell hat mehrere didaktische Vorteile:

- Die Studierenden können das Problem zunächst auf sehr abstrakter Ebene lösen.
- Das durch die Aktivitätsdiagramme modellierte Programm wird verständlicher, da abstrakte Aktionen bei entsprechender Konvention anwendungsbezogene Namen haben, die unmittelbar verständlich sind und den abstrakten Algorithmus verdeutlichen. Die Namen sind also eine Art impliziter Kommentar.
- Treten in einer der Ebenen sehr komplexe Verzweigungen und Schleifen auf, ist dies ein Indikator dafür, dass die entsprechende Aktivität vereinfacht und auf dem nächstniedrigen Abstraktionsniveau in mehrere weniger komplexe Aktivitäten aufgespalten werden sollte.

Aktivitätsdiagramme können also, insbesondere in Kombination mit weiteren UML Diagrammtypen, ein wertvolles Werkzeug zur Entwicklung konkreter Programme darstellen, weshalb ein kombinierter Aufgabentyp sinnvoll ist, bei dem den Studierenden die Ausführbarkeit von Aktivitätsdiagrammen und deren Äquivalenz zu Programmcode aufgezeigt wird.

3.2. Beschreibung des Aufgabentyps

Aus pragmatischen Gründen werden entsprechende Übungsaufgaben in der Regel separat gestaltet, so dass entweder die abstrakte Modellierung oder die Programmierung geübt und bewertet wird. Es wurde nun einen Aufgabentyp konzipiert, bei dem die Modellierung von dynamischen Programmeigenschaften mit Hilfe von Aktivitätsdiagrammen gemeinsam mit der Erstellung von Programmcode eingeübt werden kann. Die Lernziele dabei sind:

- Erstellen von syntaktisch korrekten Aktivitätsdiagrammen
- Verwendung von Bedingungen und entsprechenden Verzweigungen
- Verwendung von Schleifen
- Übergabe und Rückgabe von Parametern
- Kapselung häufig verwendeter Subroutinen
- Zugriff auf Objektattribute
- Verständnis des Zusammenhangs von Aktivitätsdiagrammen und Programmcode
- algorithmisches Problemlösen

Die konkrete Aufgabe für die Studierenden besteht darin, ein Aktivitätsdiagramm zu zeichnen, wobei sich die Syntax auf eine Untermenge der UML2 beschränkt. In bestimmten Elementen des Diagramms kann dabei allerdings konkreter Programmcode eingegeben werden, so dass das Aktivitätsdiagramm bei syntaktischer Korrektheit direkt in lauffähigen Programmcode übersetzt werden kann. Ist auch der generierte Programmcode syntaktisch korrekt, kann er ausgeführt werden, wobei den Lernenden das Ergebnis dieser Ausführung präsentiert wird. Das Ziel für die Lernenden bei dieser Aufgabe besteht also darin, ein syntaktisch korrektes Aktivitätsdiagramm zu konstruieren, dessen zugehöriger, automatisch generierter Programmcode syntaktisch und semantisch korrekt ist. Die semantische Korrektheit bezieht sich auf das in der Aufgabenstellung geforderte Verhalten des generierten Programms.

Es handelt sich bei diesem Aufgabentyp um eine offene, stark strukturierte, maschinell interpretierbare Aufgabe: Lernende müssen hier eine Lösung konstruieren, nicht selektieren. Zudem existiert keine eindeutige, korrekte Lösung der Aufgabe, da es im Allgemeinen sehr viele Möglichkeiten gibt, ein syntaktisch und semantisch korrektes Aktivitätsdiagramm zu erstellen. Die starke Strukturierung leitet sich einerseits aus der Syntax der modifizierten UML ab, andererseits aus der starken Strukturierung der formalen Sprache des Programmcodes, den bestimmte Elemente des Diagramms enthalten können. Eine syntaktisch korrekte Antwort kann dann mit Hilfe entweder eines Interpreters oder eines Compilers zusammen mit einer virtuellen Maschine maschinell interpretiert werden.

Feedback wird hier nicht summativ, sondern formativ generiert und präsentiert. Nach jedem Zwischenschritt kann den Lernenden Rückmeldung gegeben werden. Es existieren vier Feedback-Schleifen, wobei die vierte Schleife optional ist und bezüglich der Reihenfolge auch mit der dritten vertauscht werden kann.

- erste Schleife: Feedback zur Syntax des Aktivitätsdiagramms
- zweite Schleife: Feedback zur Syntax des generierten Programmcodes
- dritte Schleife: Feedback zur Semantik des generierten Programmcodes
- vierte Schleife: Feedback zu Eigenschaften des Programmcodes

Abbildung 3.1 zeigt den Prozess bis zur Lösung der Aufgabe als Flussdiagramm. Die Fehleranzeige in der ersten Schleife besteht aus einer verbalen Auflistung der vorhandenen Fehler, welche aufgrund der starken Strukturierung der Antwort maschinell generiert werden kann. Auch die Fehleranzeige der zweiten Schleife ist eine verbale Auflistung von Fehlern, welche allerdings direkt vom entsprechenden Interpreter oder Compiler generiert und den Lernenden nach optionaler Aufbereitung präsentiert wird.

Das Feedback zur Semantik des Programms in der dritten Schleife wird erstellt, indem der generierte Programmcode ausgeführt wird. Zur Bewertung der Ausführung gibt es verschiedene Möglichkeiten. Eine bekannte Methode sind Unit-Tests, bei welchen das Programm mit verschiedenen, statisch vorliegenden Parametern ausgeführt wird und die Resultate mit erwarteten Werten verglichen werden. Werden die Parameter dynamisch zufällig erzeugt, können die korrekten Resultate von einer Musterlösung bzw. Referenzimplementierung berechnet und verglichen werden. Ebenso ist es bei Verwendung entsprechender Frameworks möglich, die Ausführung des Programms zu visualisieren,

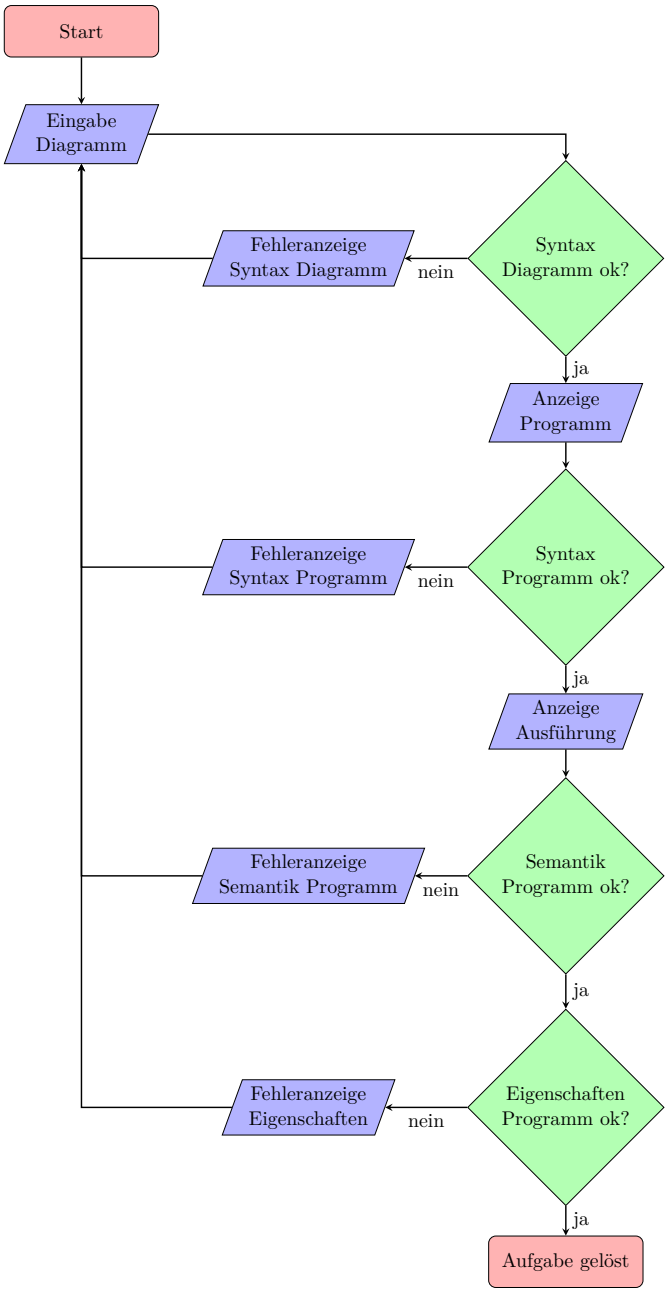


Abbildung 3.1.: Feedback-Schleifen im kombinierten Aufgabentyp zur Erstellung von UML-Aktivitätsdiagrammen mit automatischer Erzeugung von Programmcode

beispielsweise indem eine Animation berechnet und angezeigt wird. Dies kann zusätzlich zu Unit-Tests erfolgen oder auch als alleinige Rückmeldung dienen, bei der Lernende ihre Bearbeitung selbst bewerten.

In der vierten Schleife kann Feedback zu weiteren Eigenschaften des erzeugten Programmcodes gegeben werden. Beispielsweise kann das Programm auf „Code Smells“ bzw. „Bad Smells“ untersucht werden. Bei diesen Smells handelt es sich um konkrete Eigenschaften eines Programms, die ein Refactoring, also einen Umbau nahelegen. [Fowler und Beck, 2000] nennen unter anderem diese Beispiele:

- **Duplizierter Code:** Der gleiche Code befindet sich an mehr als einer Stelle.
- **Lange Methode:** Eine Routine ist zu lang.
- **Lange Parameterliste:** Einer Routine werden zu viele Parameter übergeben.
- **Neid:** Eine Methode scheint mehr an einer anderen Klasse interessiert zu sein, als an ihrer eigenen.

Fowler und Beck beziehen sich hier zwar auf objektorientierte Programmierung, doch sind diese Code Smells auch auf andere Programmier-Paradigmen anwendbar oder haben dort Entsprechungen.

In dieser Schleife kann ebenfalls überprüft werden, ob es sich bei dem Programm um ein Plagiat handelt. Ansätze und Implementierungen dazu existieren seit vielen Jahren [Bellon et al., 2007][Roy und Cordy, 2007]. Wird das Programm als Plagiat identifiziert, kann die Bearbeitung abgelehnt werden.

Natürlich wird den Lernenden auch außerhalb dieser Schleifen noch weiteres Feedback gegeben: Treten nach den Zwischenschritten keine Fehler auf, erhalten Lernende positives Feedback. Auch ist der automatisch generierte Programmcode stets ein Feedback für das eigene Aktivitätsdiagramm, unabhängig von seiner syntaktischen oder semantischen Korrektheit.

In jeder der Feedback-Schleifen wird den Lernenden Feedback der Formen KR, EF und AUC gegeben (siehe Abschnitt 1.4 auf Seite 4). Die Korrektheit der Antwort bezieht sich dabei natürlich jeweils nur auf den geprüften Aspekt, also in der ersten Schleife auf die Syntax des Diagramms. In der dritten Schleife kann zusätzlich Feedback der Form KP gegeben werden, wenn berechnet werden kann, zu welchem Grad das erstellte Programm das in der Aufgabenstellung geforderte Verhalten aufweist. Die Präsentation einer Musterlösung (Feedback der Form KCR) kann im Anschluss an eine erfolgreiche Bearbeitung erfolgen.

Der Aufwand, den die Autoren der Aufgaben speziell zur Generierung des Feedbacks betreiben müssen, unterscheidet sich in den verschiedenen Feedback-Schleifen erheblich. Für das Feedback der ersten beiden Schleifen ist auf Seiten der Autoren keine weitere Arbeit zu leisten. Das Feedback lässt sich maschinell direkt aus der Syntax von Aktivitätsdiagrammen bzw. Fehlermeldungen des Compilers oder Interpreters ableiten. Der Aufwand für das Feedback der dritten Schleife hängt vom verwendeten Ansatz ab: Sollen Unit-Tests genutzt werden, müssen für jede Aufgabe entsprechende Tests implementiert

werden. Wenn mit zufällig erzeugten Parametern getestet wird, muss auch eine Musterlösung erstellt werden, was aus didaktischen Gründen aber meist ohnehin passiert. Wird die Ausführung des Programms von einem Framework visualisiert, ist seitens des Autors keine Arbeit nötig. Ebenso verhält es sich für die Generierung des Feedbacks in der vierten Schleife, da schlechte Eigenschaften der Programm-Strukturen von Aufgaben unabhängig sind und maschinell berechnet werden können.

Auch der von Entwicklern zu betreibende Aufwand für die Implementierung der Software-Komponenten zur Feedback-Generierung unterscheidet sich je nach Schleife. Für die erste Schleife ist der Aufwand gering, da die Syntax maschinell sehr leicht geprüft werden kann. Die Feedback-Generierung der zweiten Schleife wird vom Compiler oder Interpreter übernommen, wobei eine eventuelle Aufbereitung der Fehlermeldungen zusätzlich implementiert werden muss. In der dritten Schleife ist ebenfalls die Wahl des Ansatzes ausschlaggebend: Für Vergleiche von Resultaten durch Unit-Tests bzw. anhand einer Referenzimplementierung ist der Aufwand gering. Soll die Ausführung visualisiert werden, muss dies mit relativ hohem Aufwand implementiert werden. Der Aufwand für die Feedback-Generierung der vierten Schleife hängt maßgeblich davon ab, inwieweit bestehende Systeme verwendet werden können. Tabelle 3.1 zeigt eine Übersicht des jeweils zu betreibenden Aufwands. Dabei ist natürlich zu beachten, dass vom Autor die jeweilige Arbeit bei der Erstellung jeder Aufgabe zu leisten ist, die der Entwickler nur einmalig. Zudem sind Aufwände von Autor und Entwickler natürlich nicht direkt vergleichbar.

Schleife	**Aufwand Autor**	**Aufwand Entwickler**
1. Syntax Diagramm	-	gering
2. Syntax Programm	-	- oder gering
3. Semantik Programm		
Unit-Tests	mittel	gering
Referenzimplementierung	mittel	gering
Visualisierung	-	groß
4. Eigenschaften Programm	-	mittel oder groß

Tabelle 3.1.: Aufwand für Autoren und Entwickler für die Feedback-Generierung der verschiedenen Feedback-Schleifen. „-“ steht für „kein (Aufwand)“.

3.3. Didaktische Variationen bei der Aufgabenstellung

Statt die Lernenden Aktivitätsdiagramme vollständig selbst erstellen zu lassen, können auch die folgenden didaktischen Variationen bei der Aufgabenstellung angewendet werden. Sie sollen dabei helfen, einerseits Anfängern den Einstieg im Umgang mit Aktivitätsdiagrammen zu erleichtern, andererseits übersichtliche Aktivitätsdiagramme zu erstellen.

3.3.1. Korrekturen in vollständigen Aktivitätsdiagrammen

Bei diesem Ansatz wird den Lernenden zusätzlich zu einer Aufgabenstellung ein „vollständiges“ Aktivitätsdiagramm vorgegeben, welches allerdings in einer der Feedback-Schleifen Fehler erzeugen würde. Einige Beispiele für solche Fehler sind:

- erste Schleife:
  - Zwischen zwei Knoten im Aktivitätsdiagramm fehlt die Verbindung.
  - Die Verbindung zwischen zwei Knoten ist falsch gerichtet.
  - Bei ausgehenden Verbindungen von Verzweigungsknoten fehlen die Bezeichnungen „true“ oder „false“
- zweite Schleife:
  - Innerhalb einer Aktivität wird eine nicht vorhandene andere Aktivität aufgerufen.
  - Innerhalb einer Aktivität wird eine vorhandene Aktivität mit Parametern falschen Typs aufgerufen.
  - Der konkrete Programmcode innerhalb eines Elements ist syntaktisch nicht korrekt.
- dritte Schleife (Beispiel: Aktivitätsdiagramm modelliert das Verhalten eines Roboters):
  - Zu einem bestimmten Zeitpunkt zur Laufzeit dreht sich der Roboter in die falsche Richtung.
  - Statt wie von der Aufgabenstellung verlangt, ein Rechteck zu zeichnen, wird ein Quadrat gezeichnet.
  - Statt wie von der Aufgabenstellung verlangt, eine rote Linie zu zeichnen, wird eine blaue Linie gezeichnet.
- vierte Schleife:
  - Der generierte Programmcode enthält Code Smells.

Ziel für die Lernenden ist es nun, diese Fehler zu finden und durch geringe Modifikationen[1] am vorgegebenen Aktivitätsdiagramm zu beheben. Die Lernenden können dazu die Generierung von Feedback in der jeweiligen Schleife anfordern und entsprechend der Fehlermeldungen vorgehen. Dies fördert die Fähigkeit, Aktivitätsdiagramme zu lesen und das Verständnis des Zusammenhangs des Aktivitätsdiagramms zu generiertem Programmcode und zum modellierten Verhalten. Die Komplexität ist hier deutlich geringer als bei einer vollständigen Erarbeitung eines Aktivitätsdiagramms. Gleichzeitig lässt sich die Interessantheit von Aufgaben (zumindest bei Fehlern in den ersten drei Schleifen) aber weitgehend beibehalten, da das Ziel, ein gewünschtes Verhalten zu modellieren, erhalten bleibt.

[1] Für die Behebung von Fehlern in der vierten Schleife sind für gewöhnlich umfangreichere Änderungen nötig, so dass Aufgabenstellungen diesen Typs eher für Fortgeschrittene zu empfehlen sind.

3.3.2. Aktivitätsdiagramm-Vorlagen

Auch bei diesem Ansatz müssen die Lernenden das Aktivitätsdiagramm nicht vollständig erarbeiten. Stattdessen erhalten Sie ein Aktivitätsdiagramm als Vorlage, in dem das Verhalten nicht vollständig modelliert ist, sondern eine Lücke aufweist. Wenn beispielsweise ein Aktivitätsdiagramm modelliert werden soll, das aus mehreren Aktivitäten besteht, kann hier eine Aktivität fehlen, welche von Lernenden erstellt werden muss.

- **Beispiel**: Es soll das Verhalten eines Roboters modelliert werden, der in einer Umgebung mit diskreten Feldern ein Rechteck zeichnen soll. Es sind alle nötigen Aktivitäten vorgegeben, jedoch fehlt die Aktivität `zeichneLinie`, mit der eine Linie bestimmter Länge gezeichnet werden soll.

Auch dieser Ansatz führt zu einer Verringerung der Komplexität bei gleichbleibender Interessantheit der Aufgabe, da auch hier das Ziel, ein gewünschtes Verhalten zu modellieren, erhalten bleibt. Die bereits in der Vorlage enthaltenen Aktivitäten können den Lernenden dabei als Beispiel dienen, wie Diagramme übersichtlich gestaltet werden können.

3.4. Verwandte Arbeiten

3.4.1. Systeme zur Programmierausbildung

In der Programmierausbildung sind Systeme, die eingegeben Programmcode überprüfen und entsprechendes Feedback generieren, bereits etabliert. Das System „DUESIE“ [Hoffmann et al., 2008] „ermöglicht die komplette Abwicklung des Übungsbetriebes zur Veranstaltung 'Einführung in die Informatik'“. Dabei lassen sich Aufgaben für Java und auch UML stellen. Allerdings betrifft dies nur UML Klassendiagramme.
Auch mit dem „Praktomat“ [Krinke et al., 2002] können Java-Aufgaben gestellt werden, wobei zur automatischen Auswertung funktionale Tests und automatische Stilprüfungen vorgenommen werden.

3.4.2. Code-Generierung aus UML Diagrammen

Bei dem vorgestellten Aufgabentyp werden zum Zweck der Feedback-Generierung Aktivitätsdiagramme in Programmcode übersetzt. Diese automatische Codegenerierung ist außerhalb der Programmierausbildung ein Werkzeug zur Softwareentwicklung. Es existieren einige Systeme zur Modellierung von UML-Diagrammen, mit welchen eine Codegenerierung möglich ist. Der generierte Programmcode, der das Modell bei korrekter Übersetzung perfekt abbildet, dient dann als Ausgangspunkt für die manuelle Programmierung.

[Usman und Nadeem, 2009] stellen ein Werkzeug namens „UJECTOR“ vor, das UML Klassen-, Sequenz-, und Aktivitätsdiagramme in Java Code übersetzen kann. Das Werkzeug wird mit bestehenden kommerziellen und quelloffenen Werkzeugen verglichen. Es

wird belegt, dass der generierte Programmcode funktionsfähig und verständlich ist. Ein weiteres Werkzeug, welches UML Diagramme in Java Code übersetzt, ist „Fujaba“ [Nickel et al., 2000]. Der Name ist ein Akronym für „From UML to Java and back again.“. Auch eine Rückübersetzung des modifizierten Codes in Diagramme ist damit möglich. [Gessenharter und Rauscher, 2011] stellen ebenfalls einen Ansatz zur Generierung von Programmcode aus Aktivitätsdiagrammen vor.

3.4.3. Visuelle Programmierung und Programmvisualisierung

Ausreichend abstrahiert stellt die Bearbeitung einer Aufgabe des vorgestellten Aufgabentyps eine visuelle Programmierung dar: [Myers, 1990] bezieht visuelle Programmierung auf alle Systeme, mit welchen der Benutzer ein Programm in zwei- oder mehrdimensionaler Weise spezifizieren kann[2]. Bei der Programmvisualisierung (oder Softwarevisualisierung) hingegen wird die grafische Darstellung nicht dazu benutzt, das Programm zu erstellen, sondern beispielsweise seine Ausführung zu veranschaulichen.
Der vorgestellte Aufgabentyp verwendet nach diesen Definition visuelle Programmierung, da durch zweidimensionale Aktivitätsdiagramme Programme spezifiziert werden. Auch eine Programmvisualisierung wird in der dritten Feedback-Schleife bei entsprechender Implementierung vorgenommen.

Myers beschreibt die visuelle Programmierung als pragmatisch notwendig, da Benutzer von Computern die verwendete Software aufgrund von fehlenden Fähigkeit zur Programmierung nicht nach ihren Wünschen modifizieren können. Die visuelle Programmierung sollte eine Möglichkeit sein, diese Problematik zu beheben, da eine zweidimensionale Darstellung von Programmen bei deren Verständnis helfe. Seiner Meinung nach überzeugten die damaligen Systeme zur visuellen Programmierung im Allgemeinen jedoch nicht, allerdings beschreibt er, dass einige der untersuchten Systeme in einigen Domänen Erfolge vorzuweisen haben, beispielsweise auch in der Programmierausbildung.

In der Tat sind in der Folgezeit einige Systeme zur Programmierausbildung entwickelt worden, welche visuelle Programmierung und Programmvisualisierung nutzen. Die wohl bekanntesten Systeme sind „Alice“, „Greenfoot“ und „Scratch“.

Alice ist eine um das Jahr 2000 erschienene Programmiersprache mit zugehöriger Entwicklungsumgebung, mit der das Erscheinungsbild und das Verhalten von Personen und Objekten in einer virtuellen dreidimensionalen Welt modifiziert werden kann [Cooper et al., 2000]. Ein Drag&Drop Konzept verhindert hier die Erstellung syntaktisch unkorrekter Programme. Alice wird an „hunderten Hoch- und Sekundarschulen“ eingesetzt [Fincher et al., 2010].

Einige Jahre später wurde Greenfoot [Koelling, 2010] veröffentlicht, eine Entwicklungsumgebung mit didaktischem Fokus, die darauf spezialisiert ist, interaktive, grafische Anwendungen zu erstellen. Die Programme werden hier direkt mit Java-Code entwickelt,

[2] Konventionelle textbasierte Programmierprachen werden dabei nicht als zweidimensional angesehen, da sie von Compilern oder Interpretern als eindimensionale Datenströme verarbeitet werden.

die Ausführung wird visualisiert, wobei parallel die Vererbungsstruktur der beteiligten Klassen angezeigt wird.

Im Jahr 2007 erschien Scratch [Maloney et al., 2010], eine visuelle Programmierumgebung für den schulischen Einsatz, deren Zielgruppe bei Kindern und Jugendlichen von 8 bis 16 Jahren liegt. Der Fokus liegt dabei auf dem Umgang mit Medien, also mit Bildern, Geräuschen, Musik und Videos. Scratch ist mittlerweile weit verbreitet, die Community zählt mehrere Millionen Mitglieder, von denen im Mai 2014 laut Angaben auf der offiziellen Webseite mehr als 200.000 tatsächlich in der Community aktiv waren [3].

Auch für die Programmierung von Anwendungen für mobile Endgeräte („Apps“) wurde mit dem Googles „App Inventor“ bereits eine visuelle Programmierumgebung in der Lehre eingesetzt [Wolber und Street, 2011].

3.4.4. Zusammenfassung

In der Literatur finden sich also Systeme für die Programmierausbildung, doch wird bei diesen die zuerst vorzunehmende Modellierung von Routinen durch Diagramme nicht berücksichtigt. In Literatur zeigt sich zudem, dass die Übersetzung von UML Aktivitätsdiagrammen in ausführbaren Code tatsächlich möglich ist. Auch haben sich Systeme zur visuellen Programmierung und Programmvisualisierung in der Lehre durchgesetzt, so dass die Voraussetzungen zur Implementierung eines neuartigen Systems zur automatischen Auswertung von Aktivitätsdiagrammen mit einer visualisierten Ausführung gegeben sind.

3.5. Fallstudie CaseTrain-WARP

Für eine Fallstudie wurde eine lose mit der Lernplattform Moodle[4] gekoppelte neuartige Webanwendung entwickelt [Ifland et al., 2014a], in der sich sowohl die Erstellung von Aktivitätsdiagrammen als auch indirekt die Java-Programmierung üben lassen, indem die Webanwendung diesen direkten Zusammenhang herstellt und dabei den Lernenden eine Rückmeldung über Syntax und Semantik des Aktivitätsdiagramms gibt. Die Anwendung wurde für Studienanfänger konzipiert und entstand im Rahmen des universitätsweiten Blended-Learning Projekts CaseTrain [Hörnlein et al., 2009] an der Universität Würzburg und heißt CaseTrain-WARP (**W**ürzburger **A**ktivitätsdiagram-**R**oboter-**P**rogrammierung). Die Webanwendung ist also nur organisatorisch, nicht aber konzeptionell oder technisch mit dem fallbasierten Trainingssystem CaseTrain verbunden.

Die Grundidee besteht darin, dass die Lernenden ein Programm erstellen, welches das Verhalten eines Roboters in einer virtuellen Umgebung steuert. Das Programm wird erstellt, indem Lernende ein UML-Aktivitätsdiagramm zeichnen. Ein Aktivitätsdiagramm

[3] http://scratch.mit.edu/statistics/ (August 2014)
[4] http://www.moodle.com

repräsentiert dabei eine Java-Klasse, die das Verhalten eines Roboters definiert. Innerhalb eines Aktivitätsdiagramms gibt es mindestens eine Aktivität, wobei jede Aktivität für eine Methode der Java-Klasse steht. Nach der automatischen Übersetzung des Diagramms in Java-Quellcode, wird dieser in der Roboter-Simulationsumgebung RoSE (**Ro**bot **S**imulation **E**nvironment, [Hermann, 2013]) ausgeführt. Die Veränderung einer virtuellen Umgebung durch Agenten bzw. Roboter wird durch eine Animation visualisiert und dient den Lernenden als Rückmeldung bezüglich der Semantik des erstellten Programms. Auch in einem solch konkreten Rahmen lassen sich die für Studienanfänger wichtigsten Konzepte von Aktivitätsdiagrammen und der Java-Programmierung einüben. Durch das spielerische Konzept wird eine deutliche Erhöhung intrinsischer Motivation angestrebt.

Eine konkrete Aufgabenstellung in CaseTrain-WARP wird durch ein „Szenario“ definiert. Ein Szenario besteht aus einer virtuellen Umgebung, einer Startposition der Roboter (mit Orientierung) in der Umgebung und einer Aufgabe, die der virtuelle Roboter in der Umgebung erfüllen soll. Die Umgebung ist stets ein Quadratgitter in einem rechteckigen Feld, wobei benachbarte Quadrate verbunden sein können. Quadrate sind dann Nachbarn, wenn sich ihre Seiten berühren. Ein Roboter kann sich von einem Quadrat zum anderen bewegen, wenn diese verbunden sind. Betrachtet man die Quadrate als Knoten und die Verbindungen als Kanten, handelt es sich formal also um einen ungerichteten Graphen, mit den aus dem Konzept des Quadratgitters abgeleiteten Einschränkungen (Kanten sind nur zwischen „benachbarten Knoten“ möglich). Knoten bzw. Quadrate können dabei verschiedene Zustände annehmen, beispielsweise eine Farbe. Diese Zustände können vom Szenario vorgegeben sein, aber auch von Robotern manipuliert werden. Für diese Manipulationen, für Orientierung in der Umgebung und zu Fortbewegung stehen Robotern (auch „Agenten“) einige sensorische und aktorische Befehle zur Verfügung. In WARP gibt es Einzelagenten- und kompetitive Multiagenten-Szenarios.

Ein Beispiel für ein solches Szenario könnte lauten:

- **Umgebung:** 20x20 quadratische Felder, alle benachbarten Felder verbunden
- **Startposition:** Feld links oben (Orientierung rechts)
- **Ziel:** Markierung aller Felder in grüner Farbe
- **aktorische Befehle:** „gehe ein Feld geradeaus“, „drehe um 90° nach links/rechts", „markiere das aktuelle Feld Grün“
- **sensorische Befehle:** „kann weiter geradeaus gehen?“, „Farbe des aktuellen Feldes?“

Wendet man hier das Modell der Aufgabenfelder nach [Rütter, 1982] an, so entsprechen Präsentation von Umgebung, Startposition und Befehlen dem Informationsfeld, Präsentation des Zieles dem Fragefeld und der Editor zur Erstellung des Diagramms dem Aufgabenfeld. Informations- und Fragefeld sind nicht Teil der WARP Webanwendung, sondern werden den Lernenden über die Lernplattform zusammen mit dem Link zur Webanwendung angezeigt.

Abbildung 3.2.: CaseTrain-WARP Editor: Vier Bereiche des zentralen Dialogs, der zur Erstellung der Aktivitätsdiagramme dient

3.5.1. Technische Aspekte

3.5.1.1. Grundlegendes Konzept

CaseTrain-WARP besteht im Front-End (also aus Anwendersicht) aus drei Komponenten:

- Der **Editor** ist die zentrale Komponente, in welcher Aktivitätsdiagramme erstellt und überprüft werden. (siehe Abbildung 3.2)
- In der **Arena** wird das Verhalten eines oder mehrerer Roboter mit einer Animation visualisiert (siehe Abbildung 3.10 auf Seite 91).
- Im Dialog zur **Einreichung** können Benutzer Roboter für ein Szenario einreichen, womit dieser Roboter als ihre Lösung für eine Aufgabe bzw. ein Szenario markiert wird.

3.5.1.2. Anbindung an Moodle und Authentifizierung

Die Webanwendung wird von Moodle aufgerufen und überträgt mittels HTTP GET Variablen die Moodle-Benutzerkennung, die Kennung der zu bearbeitenden Aufgabe bzw. des zu bearbeitenden Szenarios und einen persönlichen Authentifizierungsschlüssel. Dieser Authentifizierungsschlüssel wird von Moodle aus IP-Adresse des Benutzers, Benutzerkennung und einem Schlüsselwort generiert. Der Webanwendung ist dieses Schlüsselwort und der Algorithmus zur Erstellung des Authentifizierungsschlüssel bekannt. Beim Start der Webanwendung berechnet sie aus übermittelter Benutzerkennung und der IP-Adresse des Benutzers ebenfalls einen solchen und vergleicht ihn mit dem übermittelten. Sind die Schlüssel identisch, ist der Benutzer authentifiziert, andernfalls wird er abgewiesen.

Um Ergebnisse an Moodle zurückzumelden, können sich als Administratoren markierte Benutzer je Szenario eine CSV-Datei ausgeben lassen, in der entsprechende Ergebnisse notiert sind. Diese CSV Dateien sind so formatiert, dass sie in Moodle in eine entsprechende Moodle-Aktivität[5] eingelesen werden können, womit die Ergebnisse importiert werden können.

Zur Anbindungen von WARP an andere Lernplattformen ist die Implementierung nur weniger Schnittstellen nötig.

3.5.1.3. Verwendete Technologien

Die verwendeten Technologien sind clientseitig Adobe Flash, HTML5, JavaScript und CSS3, sowie serverseitig Java7 auf Apache Tomcat7 und MySQL5. In einer früheren Version der Anwendung, die auch im Experiment im Sommersemester 2013 eingesetzt wurde (siehe Abschnitt 3.5.5.1), wurden clientseitig auch Java-Applets verwendet.

3.5.1.4. Datenaustausch

Datenaustausch zwischen Client und Server findet mittels AJAX und JSON zwischen JavaScript und Java-Servlets statt. Zur Übermittlung des Aktivitätsdiagramms an den Server wird dieses in ein XML-Dokument konvertiert (Schema siehe Anhang ab Seite 155). Dieses wird serverseitig zur weiteren Verarbeitung in ein Java-Objekt umgewandelt, wobei die „Java Architecture for XML Binding“ (JAXB) verwendet wird. Der serverseitig generierte Java-Code wird in Textform, die ebenfalls serverseitig generierte RoSE Animation im JSON-Format an den Client geschickt.

3.5.1.5. Datenhaltung

Durch die Kopplung mit Moodle ist keine eigene Benutzerverwaltung nötig. Die Zuordnung von Benutzern zu sonstigen Daten (z. B. erstellten Robotern) geschieht über die Moodle-ID des Benutzers. Es werden folgende Benutzeraktionen in der angebundenen Datenbank gespeichert:

[5] https://docs.moodle.org/27/de/Aktivitäten (August 2014)

- Syntax-Prüfungen des Aktivitätsdiagramms. Dies umfasst u.a.:
  - die Benutzer-ID
  - das Aktivitätsdiagramm (als XML Text)
  - die ID des Szenarios, in dessen Kontext das Diagramm erstellt wurde
  - die Syntaxfehler (in JSON Format)
  - Datum und Uhrzeit
- Syntax-Prüfungen des generierten Programmcodes. Dies umfasst u.a.:
  - die Benutzer-ID
  - das Aktivitätsdiagramm (als XML Text)
  - die ID des Szenarios, in dessen Kontext das Diagramm erstellt wurde
  - den generierten Programmcode
  - die Syntaxfehler (in JSON Format)
  - Datum und Uhrzeit
- Erstellung eines Roboters. Dies umfasst u.a.:
  - die Roboter-ID
  - die Benutzer-ID
  - das zugehörige Aktivitätsdiagramm (als XML Text)
  - die ID des Szenarios, in dessen Kontext das Diagramm erstellt wurde
  - den generierten Programmcode
  - Datum und Uhrzeit
- Generieren und Betrachten einer Animation. Dies umfasst u.a.:
  - die Animation (in JSON Format)
  - die ID des Szenarios, in dem der Roboter ausgeführt wurde
  - die teilnehmenden Roboter
  - die Bewertung des Verhaltens der Roboter
  - Datum und Uhrzeit
- Einreichen eines Roboters. Dies umfasst u.a.:
  - die Benutzer-ID
  - die Roboter-ID
  - die Szenario-ID
  - Datum und Uhrzeit

Das Speichern dieser Daten ermöglicht eine umfangreiche statistische Auswertung der Nutzung von WARP.

3.5.1.6. Sicherheit

Benutzer der Anwendung erstellen clientseitig Programmcode, welcher serverseitig compiliert und ausgeführt wird, um die Animation zu erzeugen. Es besteht also die Gefahr, dass über die Webanwendung absichtlich oder unabsichtlich Schadcode auf den Server gespielt und ausgeführt wird. Eine Gefahr ist dabei lesender oder schreibender Zugriff auf das Dateisystem des Servers.
Dies wird in CaseTrain-WARP auf zweierlei Arten verhindert. Zum Einen läuft der Tomcat Prozess unter einem gesonderten Account, der keine Administratorenrechte hat und dessen Zugriff auf das Dateisystem auf bestimmte Ordner beschränkt ist. Des Weiteren wird der Java-Code der Benutzer in einer separaten virtuellen Maschine ausgeführt, deren Zugriffsrechte mit Hilfe des Security-Manager und einem übergebenen Policy-File sehr stark eingeschränkt sind. Dieses Konzept wird in diesbezüglich ähnlichen Systemen ebenso verwendet [Hoffmann et al., 2008].

Eine weitere Gefahr beim Ausführen von Fremdcode besteht darin, dass beispielsweise durch Endlosschleifen zu viele Systemressourcen verwendet werden, womit diese für andere Prozesse nicht mehr zur Verfügung stehen. Dies wird in CaseTrain-WARP bzw. in RoSE selbst dadurch verhindert, dass die Ausführung des Programms nach einer gewissen Rechenzeit abgebrochen wird.

3.5.2. Autorenwerkzeug

Für CaseTrain-WARP besteht kein gesondertes Autorensystem, allerdings stellt RoSE entsprechende Schnittstellen zur Implementierung bereit [Hermann, 2013]. Um ein neues Szenario zu erstellen, müssen drei Komponenten erzeugt werden.

- ***field*-Datei**: Dabei handelt es sich um eine Text-Datei, in welcher der initiale Zustand der Umgebung festgelegt wird, insbesondere die Größe der Umgebung, Verbindungen zwischen den Feldern und die Farbe jeden Feldes. Abbildung 3.3 zeigt den Inhalt einer solchen Datei und die Visualisierung der entsprechenden Umgebung. Ist die Klasse erstellt, muss RoSE mit dieser zusammen neu kompiliert werden. Szenarien sind also teilweise hart kodiert.
- **Governor-Klasse**: In dieser Java-Klasse, die von einer RoSE Governor-Basisklasse erbt, wird u.a. festgelegt, inwiefern die Umgebung nach dem Starten des Szenarios verändert wird. Dies kann beispielsweise benutzt werden, um zufällige Hindernisse zu erzeugen. Außerdem wird in dieser Klasse definiert, unter welchen Umständen ein Roboter die Aufgabe gelöst hat. Ein Beispiel für eine solche Java-Klasse ist im Anhang auf Seite 161 zu finden.
- ***properties*-Datei**: Diese Text-Datei besteht aus Schlüssel-Wert-Paaren, mit welchen das Szenario weiter parametrisiert werden kann. Sie wird unter anderem vom Governor des Szenarios zur Laufzeit ausgelesen.

```
S W W-W-W-W-W-W-W-W-W
| |   |
W W W W W-W-W-W-W-W-W
| | | |   |         |
W-W W-W W W W W W-W-W
| | | | | | | |     |
W W W W W W W W W W W
| | | | | | | | | | |
W W W W W W W W W-W-W
| | | | | | | | | | |
W W W W W-W W W W W W
| | | | | | | | | | |
W W W W W W W W W W W
| | | | |   | | |
W W W W W-W W-W W-W-W
| | | | |     |   |
W W-W W-W-W-W-W W W-R
| | | | |   | | | |
W W W W W-W W W W-W-W
```

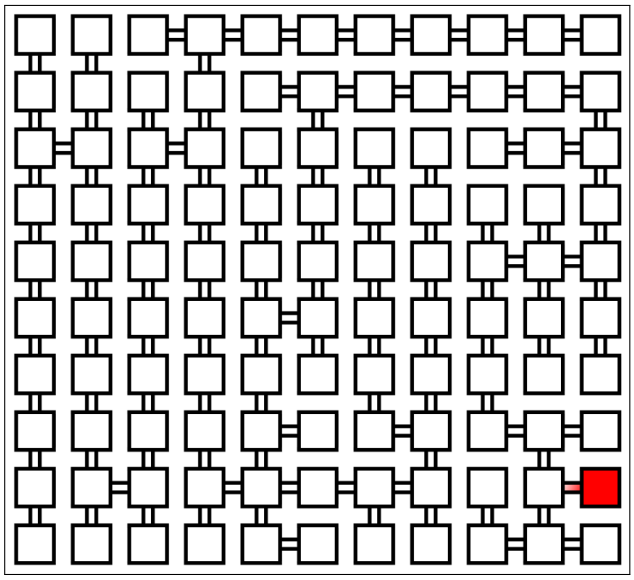


Abbildung 3.3.: Definition der Umgebung eines RoSE Szenarios: Durch die Eingabe einer Textdatei (links) in einem bestimmten Format wird die später in der Webanwendung gerenderte Umgebung (rechts) definiert. Die Buchstaben definieren die Felder und deren Farbe (`W` = weiß, `R` = rot), die Zeichen „|“ und „-“ die Verbindungen zwischen den Feldern.

Das Fehlen eines komfortablen Autorenwerkzeugs für RoSE bzw. WARP stellt sicherlich eine Einschränkung in der Gebrauchstauglichkeit dar. Es ist allerdings nicht trivial ein solches zu erstellen.

Für die Erstellung von *field*- und *properties*-Dateien könnte zwar mit wenig Aufwand ein entsprechendes Werkzeug implementiert werden. Allerdings schließt die Definition eines Governors algorithmische Definitionen mit ein und kann beliebig komplex sein. Ein Autorensystem müsste also ein entsprechendes Werkzeug beinhalten, mit dem dessen Verhalten eingegeben werden kann, beispielsweise mittels einer eigenen Skript- oder Regelsprache. Der Aufwand, in solches benutzerfreundliches Werkzeug zu implementieren, ist als sehr hoch einzuschätzen.

Zur Erstellung von Aufgaben in WARP ist derzeit also die Beteiligung eines technischen Experten notwendig.

3.5.3. Aufgabenbearbeitung

Der Dialog zur Erstellung des Aktivitätsdiagramms ist in vier Bereiche aufgeteilt (siehe Abbildung 3.2). Die Erstellung des Aktivitätsdiagramms geschieht mittels eines in Adobe Flash implementierten grafischen Editors, der bezüglich seiner Komplexität möglichst einfach gehalten wurde. Neben einigen wenigen Schaltflächen für die Konstruktion des Diagramms gibt es noch Schaltflächen zum rückgängig machen und wiederholen von Einzelschritten, und weitere zum Öffnen, Speichern (als XML-Dokument) und Exportieren (als Bild-Dokument) des Aktivitätsdiagramms von einem bzw. auf einen Datenträger, beispielsweise die Festplatte.

3.5.3.1. Syntax eines Aktivitätsdiagramms in WARP

Um Aktivitätsdiagramme maschinell weiter verarbeiten zu können, muss deren Syntax wohldefiniert werden: Ein syntaktisch korrektes Aktivitätsdiagramm in CaseTrain-WARP besteht aus einer Menge von Aktivitäten. Eine Aktivität besteht aus ihrer Signatur, einem Rückgabe-Parameter und einem gerichteten Graphen. Eine Signatur hat die gleiche Syntax wie eine Signatur einer Java-Methode.

Der gerichtete Graph G in einer Aktivität ist ein Paar (V, E) mit einer Menge V von Knoten und einer Menge von Kanten E. Jeder Knoten $v \in V$ hat einen bestimmten Knotentyp $t \in T$. Es gibt damit eine Funktion f_t , die jedem Knoten einen Typ zuordnet. Es gilt also:

$$G = (V, E) \tag{3.1}$$

$$f_t : V \to T, v \mapsto t \tag{3.2}$$

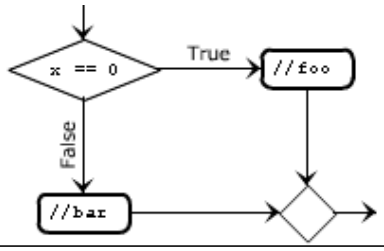
$$T = \{s, e, a, vz, vb, if, fd, dw, wd\} \tag{3.3}$$

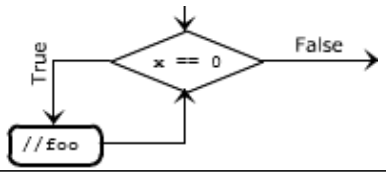
Die Knotentypen entsprechen dabei:

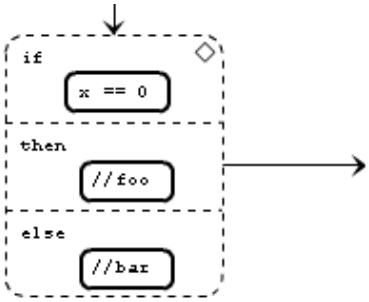
- s: Startknoten
- e: Endknoten
- a: Aktionsknoten
- vz: Verzweigungsknoten
- vb: Verbindungsknoten
- if: If-Then-Knoten
- fd: For-Do-Knoten
- dw: Do-While-Knoten
- wd: While-Do-Knoten

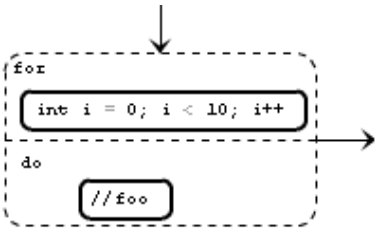
Abbildung 3.4 zeigt Beispiele dieser Knotentypen.

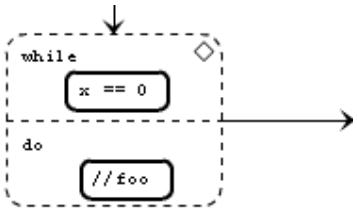
Knotentyp:	**Beispiel:**	**generierter Java-Code:**

Verzweigungsknoten und Verbindungsknoten		`if (x == 0){`
		`  //foo`
		`} else {`
		`  //bar`
		`}`

Verzweigungsknoten (als Schleife)		`while (x == 0){`
		`  //foo`
		`}`

If-Then-Knoten		`if (x == 0){`
		`  //foo`
		`} else {`
		`  //bar`
		`}`

For-Do-Knoten		`for (int i=0; i<10; i++){`
		`  //foo`
		`}`

While-Do-Knoten		`while (x == 0){`
		`  //foo`
		`}`

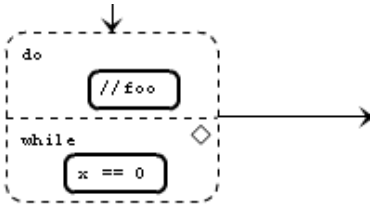
Do-While-Knoten		`do {`
		`  //foo;`
		`} while (x == 0);`

Abbildung 3.4.: Syntax und Semantik der Knotentypen eines Aktivitätsdiagramms

Jeder gerichtete Graph muss genau einen Start-und einen Endknoten enthalten. Es gilt also:

$$\exists! v \in V : f_t(v) = s \tag{3.4}$$
$$\exists! v \in V : f_t(v) = e \tag{3.5}$$

Ein- und Ausgangsgrad jeden Knotens ist 1, wobei folgende Ausnahmen gelten: Der Eingangsgrad von Startknoten und der Ausgangsgrad von Endknoten ist 0. Außerdem ist der Ausgangsgrad von Verzweigungsknoten 2, der Eingangsgrad von Verbindungsknoten ist 1 oder 2. Es gilt also:

$$\forall v \in V : f_t(v) \neq s \wedge f_t(v) \neq vb \implies d_G^-(v) = 1 \tag{3.6}$$
$$\forall v \in V : f_t(v) \neq e \wedge f_t(v) \neq vz \implies d_G^+(v) = 1 \tag{3.7}$$
$$\forall v \in V : f_t(v) = s \implies d_G^-(v) = 0 \tag{3.8}$$
$$\forall v \in V : f_t(v) = e \implies d_G^+(v) = 0 \tag{3.9}$$
$$\forall v \in V : f_t(v) = vb \implies 1 \leq d_G^-(v) \leq 2 \tag{3.10}$$
$$\forall v \in V : f_t(v) = vz \implies d_G^+(v) = 2 \tag{3.11}$$

Dabei bezeichnet $d_G^-(v)$ den Eingangsgrad, $d_G^+(v)$ den Ausgangsgrad von Knoten v.

Inhalt von Knoten und Blöcken

Aktionsknoten und Verzweigungsknoten beinhalten Text, wobei dieser auch leer sein kann. If-Then-, For-Do-, While-Do- und Do-While-Knoten bestehen aus einer Liste von Blöcken. Es existieren folgende Typen von Blöcken:

- If-Block
- Then-Block
- Else-If-Block
- For-Block
- While-Block
- Do-Block

Welche Knoten welche Blöcke in welcher Reihenfolge enthalten können, ergibt sich trivial aus der Java-Syntax der entsprechenden Kontrollstrukturen. Auf eine diesbezügliche Formalisierung wird hier aus Gründen der Übersichtlichkeit verzichtet.

Folgende Blöcke beinhalten einen einzelnen Aktionsknoten: If-Block und Else-If-Block (If-Then-Knoten), For-Block (For-Do-Knoten) und While-Block (While-Do- und Do-While-Knoten).

Then-Blöcke (If-Then-Knoten) und Do-Blöcke (For-Do-Knoten, While-Do-Knoten, Do-While-Knoten) werden als Knoten-Container bezeichnet und beinhalten einen gerichteten Graphen, für den die gleichen syntaktischen Regeln gelten, wie für den Graphen

einer Aktivität. Ausahme ist, dass sie keine expliziten Start- und Endknoten besitzen. Stattdessen müssen sie genau einen Knoten ohne eingehende und genau einen Knoten ohne ausgehende Verbindung enthalten. Diese Knoten dienen dann als implizite Start- und Endknoten des Graphen. Für diesen reduzierten Graphen G_r gilt also:

$$G_r = (V, E, v_s, v_e) \tag{3.12}$$
$$f_{t_r} : V \to T_r, v \mapsto t_r \tag{3.13}$$
$$T_r = \{a, vz, vb, if, fd, dw, wd\} \tag{3.14}$$
$$d^-_{G_r}(v_s) = 0 \tag{3.15}$$
$$d^+_{G_r}(v_e) = 0 \tag{3.16}$$
$$\forall v \in V : v \neq v_s \wedge f_{t_r}(v) \neq vb \implies d^-_{G_r}(v) = 1 \tag{3.17}$$
$$\forall v \in V : v \neq v_e \wedge f_{t_r}(v) \neq vz \implies d^+_{G_r}(v) = 1 \tag{3.18}$$
$$\forall v \in V : f_{t_r}(v) = vb \implies 1 \leq d^-_{G_r}(v) \leq 2 \tag{3.19}$$
$$\forall v \in V : f_{t_r}(v) = vz \implies 1 \leq d^+_{G_r}(v) \leq 2 \tag{3.20}$$

Dabei ist v_s der implizite Start- und v_e der implizite Endknoten des reduzierten Graphen.

Diskussion der Syntax
Da diese Anwendung für Studienanfänger konzipiert wurde, wurde in Hinblick auf die entsprechenden Lernziele (vergleiche Abschnitt 3.2) auf einige Konzepte aus der UML2 verzichtet, beispielsweise auf Parallelisierung (Splitting und Synchronisation), Objektknoten, Ausnahmebehandlung und Ereignisse.
Des Weiteren wurde die Syntax präzisiert, um die Aktivitätsdiagramme maschinell weiter verarbeiten zu können. Bei reduzierten Graphen innerhalb von Knoten-Containern (Then-Block und Do-Block) wurde auf explizte Start- und Endknoten zugunsten einer besseren Übersicht beim Zeichnen der Graphen verzichtet.

3.5.3.2. Semantik eines Aktivitätsdiagramms

Die Semantik eines syntaktisch korrekten Aktivitätsdiagramms in CaseTrain-WARP ist Java-Code. Die syntaktische Korrektheit wird vorausgesetzt, da die Semantik sonst nicht wohldefiniert werden kann. Um syntaktisch korrektem Java-Code generieren zu können, ist sie allerdings nur notwendig, nicht hinreichend, da in Aktions- und Verzweigungsknoten fehlerbehafteter Quellcode frei eingegeben werden kann. Ein Aktivitätsdiagramm wird stets in eine Java-Klasse übersetzt, wobei die Aktivitäten Methoden dieser Klasse entsprechen. Der „Rahmen“ der Klasse ist dabei durch RoSE vorgegeben: Die erstellte Klasse „WARPRobot“ erbt stets von einer Klasse „Base“, welche die Basisklasse für Roboter in RoSE ist. Außerdem enthält sie bereits die statische Methode `main()` , welche ein WARPRobot Objekt instanziiert und dessen `run()` Methode aufruft.
Dieser Klasse werden nun die im Aktivitätsdiagramm durch Aktivitäten definierten Methoden hinzugefügt. Die Signatur der Methode wird direkt aus der Signatur der Aktivität übernommen. Der Methodenrumpf wird über den Graphen der Aktivität definiert.

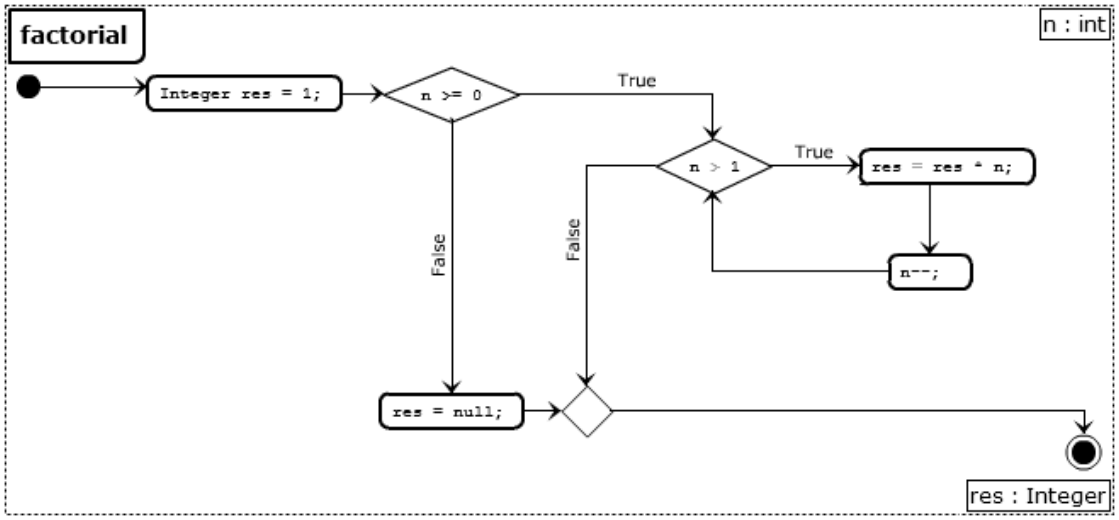


Abbildung 3.5.: Beispiel einer Aktivität zur Berechnung der Fakultät einer natürlichen Zahl

Erstellung des abstrakten Syntaxbaums

Der Graph der Aktivität wird zunächst in einen abstrakten Syntaxbaum überführt. Bei einem abstrakten Syntaxbaum handelt es sich um die Repräsentation von Quellcode als Baumgraph. Die Knoten im abstrakten Syntaxbaum repräsentieren dabei bestimmte Sprachkonstrukte, die Kanten deren Zusammensetzung. Abstrakt ist ein solcher Baum deshalb, weil bestimmte Symbole der Programmiersprache (beispielsweise Klammern) nicht explizit vorkommen, sondern durch die Konstrukte implizit definiert sind. Die Syntaxbäume, die in CaseTrain-WARP generiert werden, sind nur teilweise abstrakt, da es Aktionsknoten gibt, in welchen konkreter Java-Code steht. Dieser Code wird bei der Erstellung des Baumes nicht weiter konvertiert, sondern durch einen einzelnen „konkreten“ Knoten repräsentiert.
Abbildung 3.5 zeigt eine Aktivität zur Berechnung der Fakultät einer natürlichen Zahl, Abbildung 3.6 zeigt den zugehörigen, teilweise abstrakten Syntaxbaum. Die Wurzel des Syntaxbaumes ist stets eine „Sequenz“. Weitere Knoten im Baum ergeben sich aus den im Aktivitätsdiagramm verwendeten Knoten: While-Do-, Do-While-, und For-Do-Knoten erzeugen „Schleifen“ (unterschiedlichen Subtyps), If-Then-Knoten, und Verzweigungs- und Verbindungsknoten erzeugen „Verzweigungen“.
Eine Besonderheit stellt der aus dem Ende-Knoten des Aktivitätsdiagramms erzeugte „Ende“-Knoten im Syntaxbaum dar. Für diesen wird automatisch ein konkreter Kind-Knoten erstellt, der das Rückgabe-Statement `return res;` enthält.

Der Syntaxbaum wird nun in Java-Code umgewandelt. Das Umwandeln eines Teilbaums in Java-Code geschieht, indem abhängig von der Wurzel des Teilbaums eine bestimmte „Statement-Schablone“ erstellt wird. Ein „Verzweigung“-Knoten ergibt beispielsweise die Schablone `if (...) {...} else {...}`. Durch rekursives Aufrufen der Routine für die Teilbäume, dessen Wurzeln die Kinder des aktuellen Knotens sind, wird diese Schablone nun

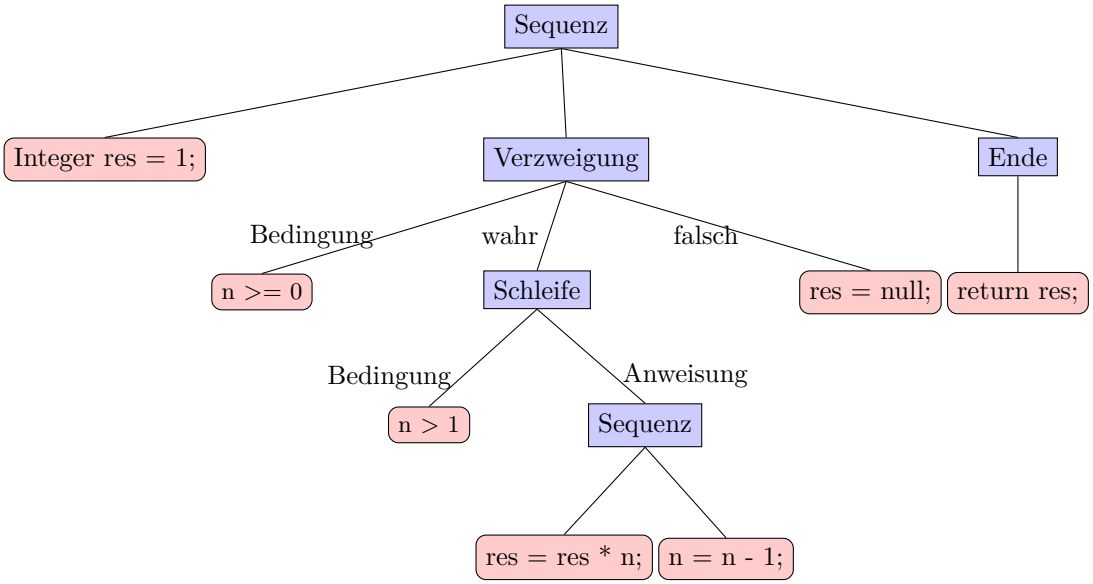


Abbildung 3.6.: teilweise abstrakter Syntaxbaum mit konkreten Knoten (rot) und abstrakten Knoten (blau), erzeugt aus der Aktivität zur Berechnung der Fakultät in Abbildung 3.5

gefüllt. Die Rekursion endet, wenn ein Teilbaum nur aus einem Blatt besteht. Blätter eines Syntaxbaums sind stets konkrete Knoten.

3.5.4. Feedback-Generierung

3.5.4.1. Feedback zur Syntax des Aktivitätsdiagramms

Grundsätzlich ist es den Lernenden möglich, mit dem Editor syntaktisch unkorrekte Diagramme zu zeichnen. Einige Aktionen, die zu syntaktisch unkorrekten Diagrammen führen, werden allerdings vom Editor verhindert. Zum Beispiel ist es nicht möglich, einem Startknoten eine eingehende Kante hinzuzufügen. Analog verhält es sich mit dem Endknoten, der nur eine eingehende Kante besitzen darf.

Auf Wunsch können Lernende ihr Aktivitätsdiagramm während der Bearbeitung auf syntaktische Korrektheit überprüfen. Das Diagramm wird daraufhin in XML umgewandelt und mittels einer AJAX-Anfrage an ein Java-Servlet geschickt. Dieses berechnet entsprechende Fehlermeldungen und liefert sie in der Antwort zurück, worauf sie als Liste angezeigt werden, ohne dass die Webseite vollständig neu geladen werden muss. Die möglichen Fehlermeldungen sind:

- Das Aktivitätsdiagramm enthält keine Aktivität.
- Eine Aktivität enthält keinen Startknoten oder keinen Endknoten. [6]

[6] Diese Fehlermeldung kann mit dem aktuell verwendeten Editor nicht vorkommen, da dieser die Eingabe entsprechender Aktivitätsdiagramme verhindert.

- Ein Knoten hat zu viele eingehende Kanten.
- Ein Knoten hat zu wenige eingehende Kanten.
- Ein Knoten hat zu viele ausgehende Kanten.
- Ein Knoten hat zu wenige ausgehende Kanten.
- Ein Knoten-Container enthält nicht genau einen Knoten ohne eingehende Kante.
- Ein Knoten-Container enthält nicht genau einen Knoten ohne ausgehende Kante.
- Eine Kante hat keinen Quell- oder keinen Zielknoten. [6]
- Bei einem Verzweigungsknoten fehlt eine ausgehende, mit „true“ gekennzeichnete Verbindung.
- Bei einem Verzweigungsknoten fehlt eine ausgehende, mit „false“ gekennzeichnete Verbindung.
- Ein Verzweigungsknoten besitzt eine ausgehende Verbindung, die weder mit „true“ noch mit „false“ gekennzeichnet ist.
- In einer Schleife gibt es mehr als einen Knoten, der eingehende und ausgehende Verbindungen zu Knoten außerhalb der Schleife hat.
- In einer Schleife gibt es keinen Knoten, der eine ausgehende Verbindungen zu einem Knoten außerhalb der Schleife hat.

Die Fehlermeldungen selbst beinhalten keinen Hinweis darauf, an welcher Stelle im Diagramm sich der Fehler befindet. Wird die Meldung jedoch mit dem Mauszeiger überfahren, wird die entsprechende Stelle im Diagramm farblich (rot) hervorgehoben (siehe Abbildung 3.7 oben).

Ist das Aktivitätsdiagramm syntaktisch korrekt, wird ein entsprechend positives Feedback gegeben (siehe Abbildung 3.7 unten) und die Lernenden haben nun die Möglichkeit, daraus Java-Code erzeugen zu lassen. Eine AJAX-Anfrage sendet das Diagramm an ein weiteres Java-Servlet, welches den Programmcode mit einigen Metadaten als Antwort liefert. Lernende können den Programmcode nun auch selbst überprüfen bzw. mit ihrem Aktivitätsdiagramm vergleichen. Überfahren sie dabei eine Zeile des Codes mit dem Mauszeiger, wird auch hier die entsprechende Stelle im Programmcode farblich (blau) hervorgehoben.

3.5.4.2. Feedback zur Syntax des Java-Codes

Ein generierter Java-Code kann jederzeit auf syntaktische Korrektheit überprüft werden. Eine AJAX-Anfrage sendet den Code und liefert eine Liste mit Fehlermeldungen, welche serverseitig von einem Java-Compiler erzeugt werden. Auch hier wird durch Überfahren der Fehlermeldung mit dem Mauscursor die entsprechende Stelle in Code und Diagramm farblich (rot) hervorgehoben (siehe Abbildung 3.7 unten). Dieses Feedback ist diesbezüglich also ähnlich dem von Entwicklungsumgebungen (Eclipse, BlueJ etc.), in denen Fehler im Programmcode ebenfalls optisch markiert und mit einer verbalisierten Fehlermeldung versehen sind.

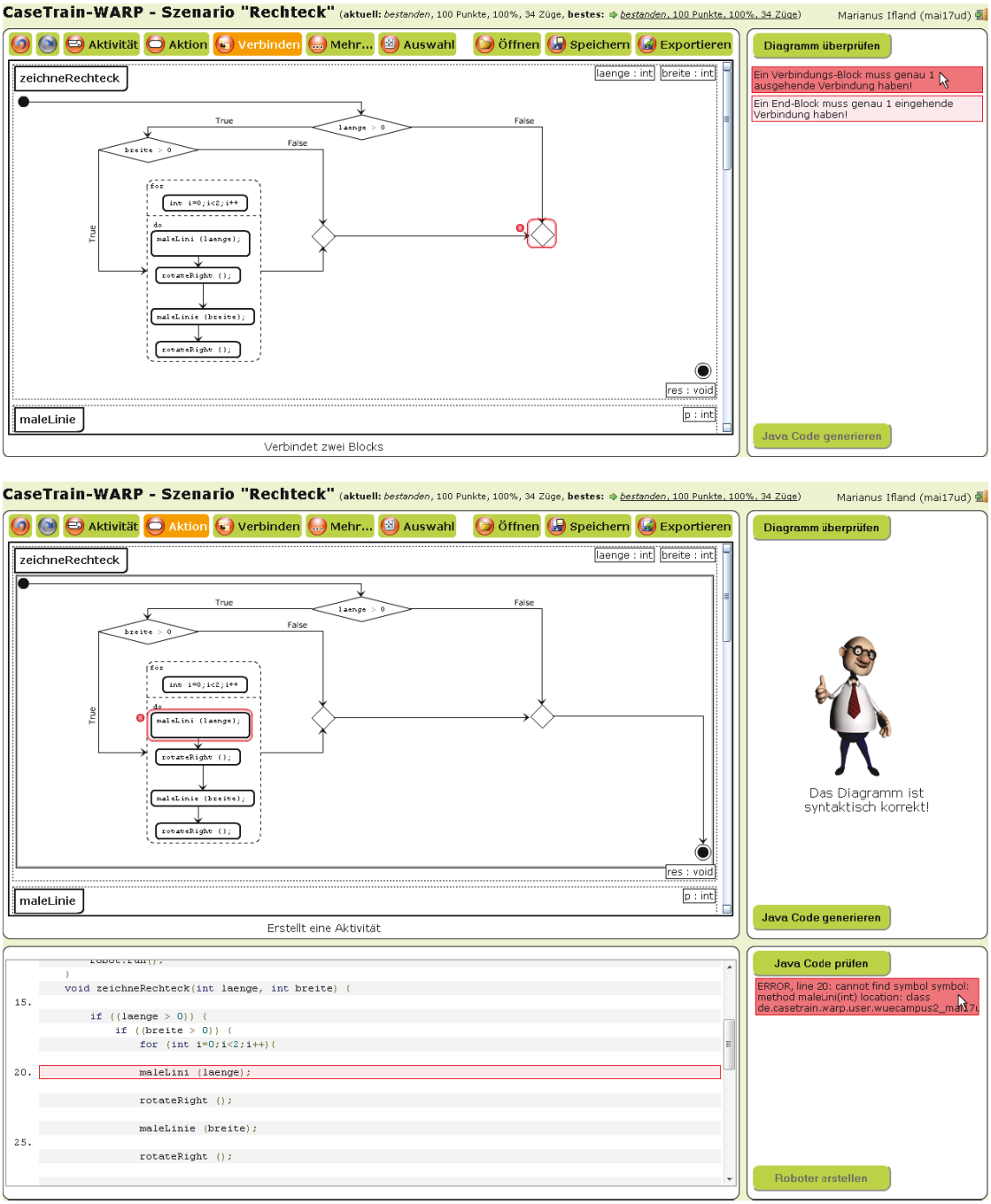


Abbildung 3.7.: CaseTrain-WARP Editor: Feedback-Generierung bezüglich der Syntax des Aktivitätsdiagramms (oben) und des generierten Programmcodes (unten). Beim Überfahren der jeweiligen Fehlermeldung mit dem Mauszeiger wird die entsprechende Stelle im Diagramm farblich hervorgehoben.

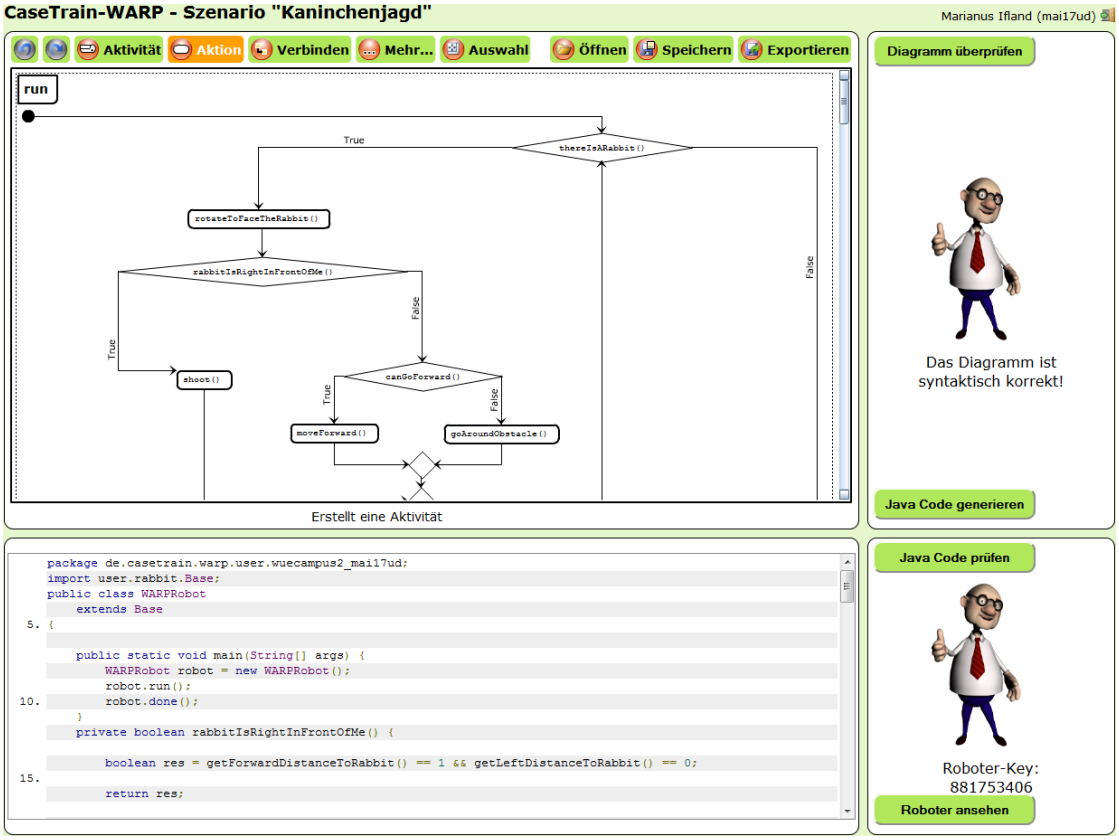


Abbildung 3.8.: CaseTrain-WARP Editor: Ist auch der Programmcode syntaktisch korrekt, kann ein Roboter mit einem entsprechenden Key generiert und in der Arena angesehen werden.

Bei syntaktisch korrektem Code wird den Lernenden analog zur syntaktischen Korrektheit des Diagramms ebenfalls ein positives Feedback angezeigt. Nun können Lernende aus dem Diagramm einen virtuellen Roboter erzeugen, worauf sie zu dessen Identifikation einen Roboter-Key erhalten (siehe Abbildung 3.8). Diesen können sie notieren, um den Roboter auch zu einem späteren Zeitpunkt in der Arena ausführen zu können. Beispielsweise für Mehrspielerszenarien können Lernende diese Keys untereinander austauschen, um die entsprechenden Roboter in der Arena gegeneinander antreten zu lassen.

Ist der Roboter und der entsprechende Key erstellt, kann direkt die Arena geöffnet werden, wobei der Key des soeben erstellten Roboters und das zuvor bearbeitete Szenario im Formular bereits vorausgewählt ist (siehe Abbildung 3.9).

Starten Lernende die Erzeugung der Animation, wird serverseitig eine RoSE-Animation erstellt. RoSE instanziiert dazu das Szenario und ruft die Methode main() des gewählten Roboters auf. Der Roboter verändert nun schrittweise den Zustand der Umgebung (z. B. durch Einfärben eines Feldes) und den eigenen Zustand (z. B. durch Veränderung der

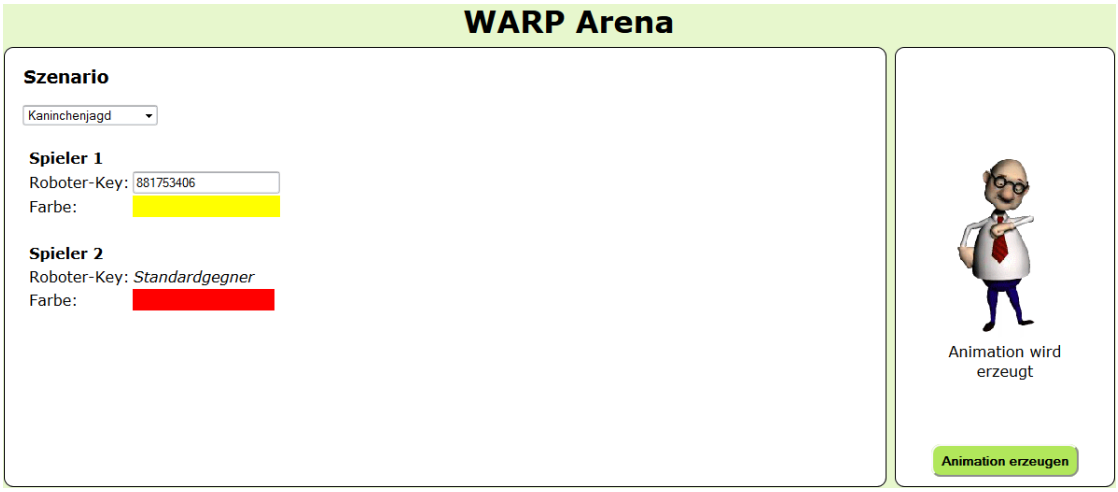


Abbildung 3.9.: CaseTrain-WARP Arena: In der Arena wird das Szenario und der auszuführende Roboter ausgewählt, wobei das Formular bei einem Aufruf aus dem Editor vorausgefüllt ist. Hier kann nun die Animation erzeugt werden.

Position). Diese Zustände der Umgebung und des Roboters werden in einem internen Format gespeichert und ergeben als Sequenz die RoSE-Animation.

3.5.4.3. Feedback zur Semantik des Java-Codes

Die Animation kann nun im RoSE-Player in der Arena betrachtet werden, wobei die einzelnen Zustände der Umgebung und des Roboters gerendert werden. Den Lernenden ist es nun möglich, das Verhalten ihres Roboters durch wiederholtes, explizites Aufrufen des nächsten Zustandes der Umgebung zu betrachten. Alternativ kann die Animation auch automatisch abgespielt werden. Lernende können so sehen, ob der durch ihr Aktivitätsdiagramm erstellte Roboter sich korrekt, also entsprechend der Aufgabenstellung des Szenarios verhält und diese löst oder nicht. Außerdem erfolgt eine kurze verbale Rückmeldung über Erfolg oder Misserfolg (siehe Abbildung 3.10). In Multiagenten-Szenarien wird hier auch Rückmeldung darüber gegeben, welcher Spieler das Spiel gewonnen hat.

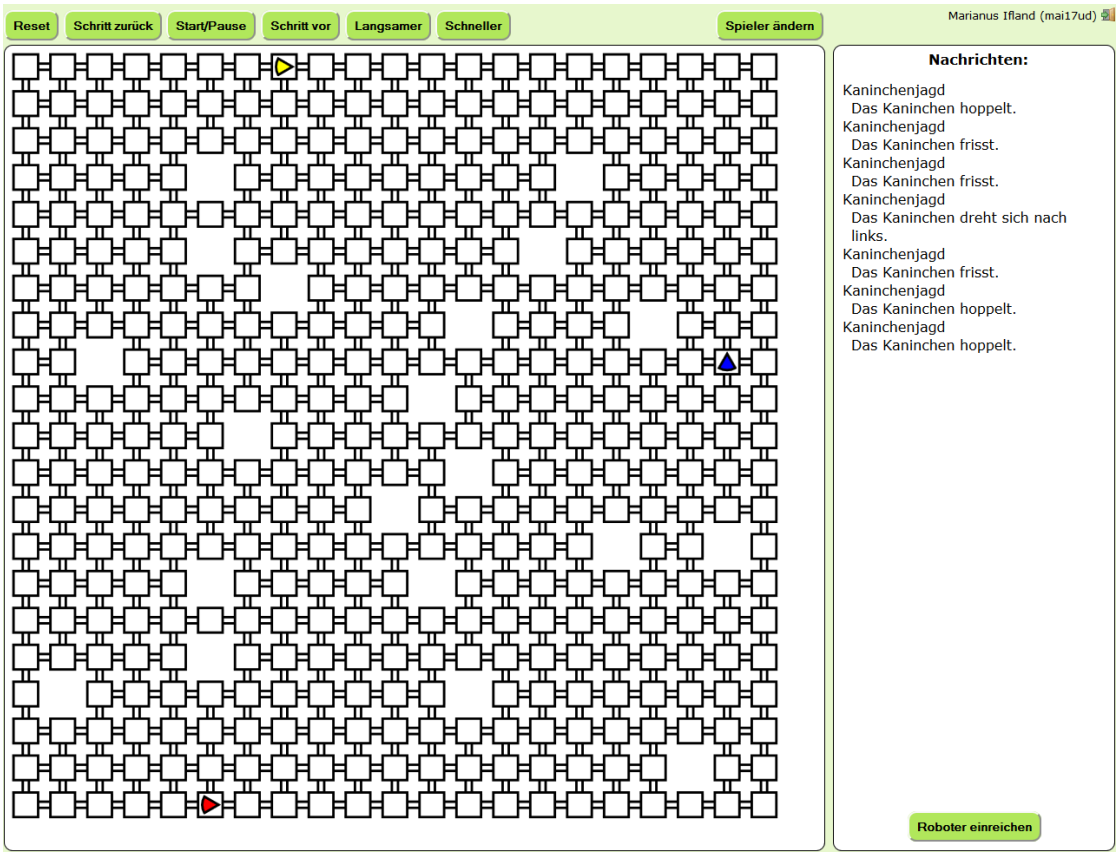


Abbildung 3.10.: CaseTrain-WARP Arena: Die Abbildung zeigt einen Zustand der Umgebung eines Multiplayer-Szenarios, in welchem die Aufgabe für die beiden Spieler (rot und gelb) darin besteht, sich zu einem sich zufällig bewegenden Kaninchen (blau) zu bewegen und es dann zu erlegen. Auf der rechten Seiten werden Statusmeldungen angezeigt.

3.5.5. Evaluation und Ergebnisse

Das Trainingssystem WARP wurde in den Sommersemestern 2013 und 2014 im Übungsbetrieb zur Vorlesung Softwaretechnik an der Universität Würzburg zur Erstellung von Aktivitätsdiagrammen eingesetzt. Die Rahmenbedingungen waren dabei identisch zum Experiment zur Bewertung von Klassendiagrammen (siehe Abschnitt 2.6 auf Seite 46). Zur Bearbeitung der Aufgaben in WARP hatten die Studierenden jeweils etwa eine Woche Zeit. Aus der Vorlesung hatten sie theoretische Vorkenntnisse zur Erstellung von Aktivitätsdiagrammen, allerdings hatten sie keine Vorkenntnisse zur der Verwendung von WARP. Sie kannten lediglich den eingesetzten Editor aus den Aufgaben zur Erstellung von Klassendiagrammen (siehe Kapitel 2) aus der Vorwoche, wobei nur sehr wenige Funktionen der Editoren (Rückgängig machen & Wiederherstellen, Verschieben

von Elementen) identisch, die meisten also auf die Art des zu erstellenden Diagramms angepasst sind.

3.5.5.1. Experiment im Sommersemester 2013

Im Sommersemester 2013 wies WARP folgende Unterschiede gegenüber der zuvor vorgestellten Version auf:

- In den Aktivtätsdiagrammen waren keine Zykel erlaubt. Schleifen mussten also mit strukturierten Knoten abgebildet werden.
- Der Aktivitätsdiagramm-Editor verhinderte einige syntaktisch fehlerhafte Eingaben, wie z. B. mehrere eingehende Verbindungen bei Aktionsknoten.
- In der Arena wurde ein Java-Applet verwendet (statt Standard-Webtechnologie).
- Es gab keine expliziten Einreichungen (siehe unten).

Es wurden drei Szenarien bearbeitet:

- Szenario „Rechteck“:
  - Umgebung: 21x21 Felder, alle benachbarten Felder verbunden
  - Startposition: mittleres Feld (Blick nach rechts)
  - Aufgabe: „Zeichnen Sie ein Rechteck der Seitenlänge mit Breite/Höhe 4/5“
- Szenario „Hoover“:
  - Umgebung: 12x8 Felder, alle benachbarten Felder verbunden
  - Startposition: linkes oberes Feld (Blick nach rechts)
  - Aufgabe: „Markieren Sie alle Felder in einer beliebigen Farbe“
- Szenario „Labyrinth“:
  - Umgebung: 11x10 Felder, nicht alle Felder sind verbunden (statisches Labyrinth), ein Feld ist farblich markiert
  - Startposition: linkes oberes Feld (Blick nach rechts)
  - Aufgabe: „Finden Sie das farblich markierte Feld“

Die Aufgaben waren dabei ausführlicher und bildhafter formuliert als hier angegeben. Im Szenario „Hoover“ soll der ausführlichen Beschreibung nach ein Staubsauger modelliert werden, der alle Felder der Umgebung saugt. Bei der Aufgabe „Labyrinth“ ist das Ziel, in einem Labyrinth nach einem Schatz zu suchen (vollständige Aufgabenstellung siehe Anhang ab Seite 176).

Den Lernenden standen dazu eine Reihe von aktorischen und sensorischen Befehlen zur Verfügung, Tabelle 3.3 zeigt eine entsprechende Übersicht.

Die Studierenden konnten alle Szenarien innerhalb eines Zeitraumes von einer Woche beliebig oft bearbeiten. Die CaseTrain-WARP-Webanwendung speicherte während dieses Experiments jedes Aktivitätsdiagramm nach jeder Änderung in einer Datenbank ab,

Rückgabe	**Befehl**	**Bedeutung**
void	moveForward()	gehe ein Feld in Blickrichtung
void	rotateRight()	drehe dich nach links
void	rotateLeft()	drehe dich nach rechts
void	moveRandom()	gehe mit 50 % Wahrscheinlichkeit ein Feld vor, ansonsten drehe dich mit jeweils 25 % Wahrscheinlichkeit in eine Richtung und gehe dann ein Feld vor
void	markCurrentNode(Color c)	markiere das aktuelle Feld mit Farbe c
bool	canGoForward()	kann ich in Blickrichtung gehen?
bool	canGoLeft()	kann ich nach links gehen?
bool	canGoRight()	kann ich nach rechts gehen?
bool	isMarked()	ist der Knoten, auf dem ich gerade stehe, mit einer Farbe markiert?

Tabelle 3.3.: Aktorische und sensorische Befehle, die den Lernenden für ihren Roboter in den Szenarien „Rechteck“, „Hoover“ und „Labyrinth“ zur Verfügung standen

	Rechteck	**Hoover**	**Labyrinth**	**gesamt**
Anzahl Bearbeitungen	244	228	206	678
Anteil erfolgreicher Bearbeitung	91,8 %	91,7 %	87,3 %	90,4 %
Ø Anzahl Aktivitäten	2,9	3,0	2,4	2,8
Ø Anzahl Knoten	12,8	20,3	13,1	15,4
Ø Anzahl Kanten	11,6	16,3	11,0	13,0

Tabelle 3.4.: Größe der Stichproben sowie die Erfolgsquote der Bearbeitungen der drei bearbeiteten Szenarien im Sommersemester 2013

sofern es syntaktisch korrekt und auch der generierte Java-Code syntaktisch korrekt war. Es gab also keine expliziten Einreichungen. Auf diese Weise wurden über 12.000 Aktivitätsdiagramme von 255 verschiedenen Studierenden in der Datenbank gespeichert. Daraus wurde nun für jede Aufgabe eine Teilmenge gebildet, welche jeweils die „finalen Bearbeitungen“ der Studierenden für diese Aufgabe beinhaltet. Die finale Bearbeitung eines Studierenden bei einer Aufgabe ist das zuletzt erstellte Diagramm, welches zu einer erfolgreichen Lösung der jeweiligen Aufgabe führte bzw. bei Studierenden, die die Aufgabe nicht lösen konnten, das zuletzt erstellte Diagramm. Die Teilmengen beinhalten also für jeden Studierenden genau eine Bearbeitung pro Aufgabe.
Tabelle 3.4 zeigt die Größe der Teilmengen, die Erfolgsquote der Bearbeitungen, sowie weitere Statistiken über die enthaltenen Graphen. Die Tabellen 3.5 und 3.6 zeigen die durchschnittliche Benutzung der Knotentypen und der vorgegebenen aktorischen und sensorischen Befehle.

	Rechteck	**Hoover**	**Labyrinth**	**gesamt**
Aktionsknoten	9,1/100 %	15,7/100 %	8,9/100 %	11,3/100 %
Verzweigungsknoten	0,7/57 %	0,5/29 %	1,0/39 %	0,7/42 %
Verbindungsknoten	0,7/55 %	0,5/27 %	1,0/39 %	0,7/41 %
If-Then-Knoten	0,2/15 %	1,2/53 %	1,0/51 %	0,8/39 %
For-Do-Knoten	2,1/97 %	0,3/18 %	0,1/5 %	0,9/42 %
While-Do-Knoten	0,1/4 %	1,9/90 %	1,0/82 %	1,0/56 %
Do-While-Knoten	0,0/2 %	0,2/14 %	0,0/2 %	0,1/6 %

Tabelle 3.5.: Nutzung der Knoten nach Typ im Sommersemester 2013: Der erste Wert gibt den Mittelwert der in den Diagrammen enthaltenen Knoten des entsprechenden Typs an, der zweite Wert gibt den Anteil an Diagrammen an, die mindestens einen Knoten entsprechenden Typs beinhalten.

	Rechteck	**Hoover**	**Labyrinth**	**gesamt**
moveForward()	1,5/99 %	3,5/99 %	3,7/96 %	2,8/98 %
rotateRight()	0,7/30 %	2,3/95 %	1,8/92 %	1,6/71 %
rotateLeft()	1,5/70 %	2,0/93 %	1,7/86 %	1,8/83 %
moveRandom()	0,0/0 %	0,0/1 %	0,1/7 %	0,0/2 %
markCurrentNode()	1,5/98 %	3,3/98 %	0,1/7 %	1,7/71 %
canGoForward()	0,0/4 %	1,8/91 %	0,9/59 %	0,9/49 %
canGoLeft()	0,0/8 %	0,3/24 %	0,5/30 %	0,2/17 %
canGoRight()	0,0/0 %	0,4/29 %	0,4/31 %	0,3/19 %
isMarked()	0,0/1 %	0,5/31 %	1,0/89 %	0,5/38 %

Tabelle 3.6.: Nutzung der vorgegebenen aktorischen und sensorischen Befehle im Sommersemester 2013: Der erste Wert gibt den Mittelwert der in den Diagrammen enthaltenen Befehle, der zweite Wert gibt den Anteil an Diagrammen an, in denen der Befehl mindestens einmal vorkommt.

In Tabelle 3.4 lässt sich erkennen, dass die Anzahl der von den Studierenden modellierten Aktivitäten nur gering schwankt. Auffällig ist allerdings die hohe Anzahl an Knoten bei der Aufgabe „Hoover“. Eine mögliche Ursache für die hohen Erfolgsquoten ist die Tatsache, dass Studierende beliebig viele Versuche zur Lösung der Aufgabe hatten.

Wie in Tabelle 3.5 zu sehen ist, ist der mit Abstand am häufigsten verwendete Knotentyp der Aktionsknoten, welcher in jedem eingereichten Diagramm vorhanden. Interessant erscheint auch die unterschiedliche Nutzung der Möglichkeiten zur Modellierung von Verzweigungen. Diese können mit Verzweigungs- und Verbindungsknoten oder mit strukturierten If-Then-Knoten modelliert werden. Beim Szenario „Rechtweck“ wurden überwiegend Verzweigungsknoten genutzt, beim Szenario „Hoover“ dagegen überwiegend If-Then-Knoten. Gleichmäßig waren sie beim Szenario „Labyrinth“ verteilt.

Ein aus Tabelle 3.6 ablesbares, auffälliges Ergebnis ist, dass die Studierenden ein Rechteck offenbar lieber durch Rotationen nach links, als durch Rotationen nach rechts zeich-

nen lassen. Ebenfalls ist auffällig, dass im Szenario „Hoover“ offenbar nur knapp ein Drittel der Studierenden durch Verwendung des Befehls isMarked() die Felder dahingehend untersucht hatten, ob der Roboter dort schon „gesaugt“ hatte.

Etwa zwei Wochen nach Bearbeitung der Aufgaben wurde den Studierenden ein schriftlicher Fragebogen vorgelegt (siehe Anhang Seite 181), in welchem auch konkrete Fragen zu CaseTrain-WARP gestellt wurden. Die Studierenden sollten bestimmte Aspekte der Anwendung mit Schulnoten (1-6) bewerten. Tabelle 3.7 zeigt die Ergebnisse dieser Umfrage.

Aspekt	**Ø Note**
Konzept des Tools, unabhängig von der Implementierung	2,28
Umsetzung des Tools, Gesamtnote	2,79
Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)	2,76
Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)	3,81
Aspekt Inhalt der Übungsaufgaben	2,19
Aspekt Fairness der Bewertung der Übungsaufgaben	2,33
Aspekt hilfreiche Erklärungen als Text oder nützliches Feedback	2,81

Tabelle 3.7.: Umfrageergebnisse zu CaseTrain-WARP im Sommersemester 2013; die verschiedenen Aspekte wurden mit Schulnoten (1-6) bewertet (n=240)

3.5.5.2. Experiment im Sommersemester 2014

Im Sommersemester 2014 wurde WARP in der Vorlesung mit einem kommentierten Video kurz vorgeführt. Außerdem standen den Studierenden eine Kurzanleitung in Textform (siehe Anhang ab Seite 195) und zwei Screencasts zu Verfügung, in welchen die grundsätzliche Funktionsweise von WARP erklärt wurde. Es wurden fünf Szenarien in WARP bearbeitet. „Rechteck“ und „Hoover“ wurden unverändert gegenüber dem Vorjahr gestellt. Zusätzlich wurde das Szenario „HooverRandom“ bearbeitet, welches eine um stochastische Elemente erweiterte Variante von „Hoover“ ist. Das „Labyrinth“ wurde ebenfalls um ein stochastisches Element erweitert und als „LabyrinthRandom“ angeboten. Erstmals wurde mit der „Kaninchenjagd“ ein Mehrspieler-Szenario gestellt, in dem die Roboter der Studenten gegen einen Standardgegner antreten mussten (vollständige Aufgabenstellung siehe Abschnitt A.2.4.2 auf Seite 179). Für die Aufgabe „Kaninchenjagd“ stand eine alternative Bearbeitungsmöglichkeit zur Auswahl (direkte Eingabe von Java-Code in einem webbasierten Editor). Es folgt eine Beschreibung der Aufgaben:

- Szenario „Rechteck“ (siehe Abschnitt 3.5.5.1)
- Szenario „Hoover“ (siehe Abschnitt 3.5.5.1)
- Szenario „HooverRandom“:
  - Umgebung: 12x8 Felder, alle benachbarten Felder verbunden; drei zufällig verteilte Felder, die zu keinem ihrer Nachbarn verbunden sind; diese Hindernisse konnten nicht am Rand und nicht nebeneinander liegen

  - Startposition: linkes oberes Feld (Blick nach rechts)
  - Aufgabe: „Markieren Sie alle Felder in einer beliebigen Farbe“
- Szenario „LabyrinthRandom“:
  - Umgebung: 11x10 Felder, nicht alle Felder sind verbunden (statisches Labyrinth)
  - Startposition: zufällig
  - Aufgabe: „Finden Sie das farblich markierte Feld“
- Szenario „Kaninchenjagd“:
  - Umgebung: 21x21 Felder, alle benachbarten Felder verbunden; 16 zufällig verteilte Felder, die zu keinem ihrer Nachbarn verbunden sind; diese Hindernisse können nicht am Rand und nicht nebeneinander liegen; in der Mitte rechts befindet sich ein Roboter (Kaninchen), der sich zufällig bewegt
  - Startposition: Spieler 1: linkes oberes Feld (Blick nach rechts), Spieler 2: linkes unteres Feld (Blick nach rechts)
  - Aufgabe: „Finden und erlegen Sie das Kaninchen“

Es standen den Studierenden wieder alle in Tabelle 3.3 aufgeführten aktorischen und sensorischen Befehle zur Verfügung, für die „Kaninchenjagd“ zusätzlich die Befehle aus Tabelle 3.8.

Rückgabe	**Befehl**	**Bedeutung**
boolean	thereIsARabbit()	lebt das Kaninchen noch?
int	getForwardDistanceToRabbit()	gebe zurück, wie viele Felder das Kaninchen vor mir steht
int	getBackwardDistanceToRabbit()	gebe zurück, wie viele Felder das Kaninchen hinter mir steht
int	getLeftDistanceToRabbit()	gebe zurück, wie viele Felder das Kaninchen links von mir steht
int	getRightDistanceToRabbit()	gebe zurück, wie viele Felder das Kaninchen rechts von mir steht
void	shoot()	erlegt das Kaninchen, wenn es direkt vor mir steht

Tabelle 3.8.: Sensorische und aktorische Befehle, die den Lernenden für ihren Roboter in dem Szenario „Kaninchenjagd“ zur Verfügung standen

Die Aufgaben konnten dabei innerhalb von zehn Tagen beliebig oft bearbeitet werden. Die Lernenden konnten also beliebig viele Roboter erstellen. Dabei wurden die erstellten Diagramme (im Gegensatz zum Experiment im Sommersemester 2013) nicht implizit nach jeder Änderung gespeichert, sondern nur dann, wenn von Lernenden eine explizite Überprüfung der Syntax angefordert wurde. Außerdem wurde der generierte Programmcode ebenfalls nur dann gespeichert, wenn für diesen eine explizite Überprüfung der

	RE	**HO**	**HOR**	**LAB**	**KAN**	**gesamt**
Anzahl Syntaxprüfungen	5192	7406	6727	2838	2541	24.704
Anzahl verschiedener Diagramme	4059	5555	5301	2288	2028	19.231
Anzahl verschiedener Benutzer	271	260	239	237	128	280
Ø Anzahl Fehler	0,65	0,80	0,56	0,44	0,46	0,62
Ø Anzahl Fehler (nur Syntaxprüfungen mit Fehlern)	3,8	2,3	3,0	2,2	2,9	2,8
Anteil Prüfungen ohne Fehler	83 %	66 %	82 %	80 %	84 %	77 %

Tabelle 3.9.: Statistiken zu den expliziten Syntaxprüfungen der Aktivitätsdiagramme im Sommersemester 2014 (RE = Rechteck, HO = Hoover, HOR = HooverRandom, LAB = LabyrinthRandom, KAN = Kaninchenjagd)

Syntax angefordert wurde. Die Lernenden konnten ihre Roboter beliebig oft in der Arena testen. Waren sie mit dem Verhalten eines Roboters zufrieden, konnten sie ihn explizit als ihre Lösung für dieses Szenario einreichen. Bei einer Einreichung wurde der Roboter erneut im entsprechenden Szenario ausgeführt und das zugehörige Ergebnis gespeichert.

Es wurden insgesamt über 24.700 Syntax-Prüfungen für über 19.200 verschiedene Aktivitätsdiagramme (erste Feedback-Schleife) und über 16.700 Syntax-Prüfungen für den Programmcode von über 13.900 verschiedenen Roboter-Klassen (zweite Feedback-Schleife) von etwa 280 Benutzern angefordert. Die Tabellen 3.9 und 3.10 zeigen Statistiken zu den Syntax-Überprüfungen der Aktivitätsdiagramme.

Auffällig ist dabei, dass beim Szenario „HooverRandom“ in den Aktivitätsdiagrammen offenbar weniger syntaktische Fehler gemacht wurden als bei den anderen Szenarien. Die niedrigere Anzahl an Studierenden bei der „Kaninchenjagd“ resultiert aus der zusätzlichen Möglichkeit, den Programmcode für den Roboter direkt einzugeben, von der 98 Studierende Gebrauch gemacht hatten.

Für die Statistiken in Tabelle 3.10 wurden alle Syntaxfehler in übergeordnete Typen gruppiert. Bei Fehlern des Typs SUB wurde die syntaktische Vorgabe verletzt, dass Subgraphen in Block-Containern genau einen Knoten ohne eingehende und genau einen Knoten ohne ausgehende Verbindung enthalten (siehe Abschnitt 3.5.3.1) müssen. Der Fehlertyp SCH schließt alle Fehler ein, die aus unkorrekter Konstruktion von Schleifen mittels Verzweigungsknoten resultieren. Bei Fehlern des Typs AVV fehlen bei Verzweigungsknoten ausgehende, mit „true“ und „false“ gekennzeichnete Verbindungen.

Der mit Abstand häufigste Fehlertyp war eine fehlerhafte Anzahl an ein- und ausgehenden Verbindungen. Eine mögliche Ursache für diese Tatsache ist, dass die anderen Fehlertypen in vielen der überprüften Aktivitätsdiagramme nicht vorkommen konnten, da entsprechende Strukturen dort nicht verwendet wurden.

Die Tabelle 3.11 und 3.12 zeigen Statistiken zu den Syntaxprüfungen der Programmcodes der automatisch generierten Roboter-Klassen. Dabei sind in Tabelle 3.12 nur die

Fehlertyp	**Häufigkeit**	
AVE (fehlerhafte Anzahl ein-/ausgehender Verbindungen)	10211	67 %
SUB (fehlerhafter implizite Start- und Endknoten in Subgraph)	2183	14 %
SCH (fehlerhafte Schleife)	2031	13 %
AVV (fehlerhafte ausgehende Verbindungen bei Verzweigungsknoten)	927	6 %

Tabelle 3.10.: Verteilung der Syntaxfehler in Aktivitätsdiagrammen nach Typ

häufigsten 7 aller 35 vorkommenden Fehlertypen aufgeführt, welche insgesamt etwa 86 % aller aufgetretenen Fehler ausmachen. Die 28 sonstigen Fehlertypen haben einen Anteil von jeweils weniger als 5 % an allen Fehlern und wurden nicht näher untersucht. Die möglichen Ursachen der 7 häufigen Fehlertypen werden im Folgenden genauer dargestellt:

- **SYE (Symbol erwartet)**: Die Ursache war in den meisten Fällen eine fehlende öffnende oder schließende Klammer oder ein fehlendes Semikolon.
- **SYN (Symbol nicht gefunden)**: Hier wurden hauptsächlich Variablen aufgerufen, die zuvor nicht definiert wurden.
- **UAU (ungültiger Ausdruck)** und **UAN (ungültige Anweisung)**: Ein Ausdruck oder eine Anweisung im Programmcode konnte vom Compiler nicht geparst werden. Die Ursachen waren dabei sehr vielfältig und wurden nicht näher untersucht.
- **MEN (Methode existiert nicht)**: Es wurde eine Methode aufgerufen, die nicht existiert. Dies schließt sowohl selbst definierte Methoden als auch Systemmethoden (Tabellen 3.3 und 3.8) mit ein. Ursache ist meist ein fehlerhaft geschriebener Methodenname beim Aufruf.
- **KIE (Klasse, Interface oder Enum erwartet)**: Bei Fehlern des Typs KIE wird die Definition einer solche Struktur erwartet, also die einer Klasse, eines Interfaces oder einer Enumeration. Dieser Fehler tritt in der Praxis vor allem bei falscher Klammersetzung auf, z. B. wenn eine überzählige schließende geschweifte Klammer eingegeben wurde.
- **MUP (Methodenaufruf mit ungültigen Parametern)**: Es wurde eine existierende Methode aufgerufen, allerdings mit ungültigen Parametern. Dies betrifft sowohl eine falsche Anzahl an Parametern, als auch unpassende Typen der Parameter.

Einige dieser Fehler erscheinen bei automatischer generiertem Programmcode zunächst ungewöhnlich. Doch konnten Lernende in Aktionsknoten beliebigen Java-Code eingeben, so dass hier theoretisch jeder in Java erzeugbare Fehler auch erzeugt werden konnte.

	RE	**HO**	**HOR**	**LAB**	**KAN**	**gesamt**
Anzahl Syntaxprüfungen	3644	4108	4812	2170	2051	16.785
Anzahl verschiedener Klassen	3024	3619	4093	1776	1621	13.970
Anzahl verschiedener Benutzer	268	253	234	234	122	274
Ø Anzahl Fehler	0,38	0,34	0,20	0,12	0,56	0,31
Ø Anzahl Fehler (nur Syntaxprüfungen mit Fehlern)	2,6	3,0	2,4	2,9	5,6	3,0
Anteil Prüfungen ohne Fehler	85 %	88 %	92 %	96 %	90 %	90 %

Tabelle 3.11.: Statistiken zu den expliziten Syntaxprüfungen der Programmcodes der automatisch generierten Roboter-Klassen im Sommersemester 2014 (RE = Rechteck, HO = Hoover, HOR = HooverRandom, LAB = LabyrinthRandom, KAN = Kaninchenjagd)

Fehlertyp	**Häufigkeit**	
SYE (Symbol erwartet)	8438	36 %
SYN (Symbol nicht gefunden)	3372	15 %
UAU (ungültiger Ausdruck)	2301	10 %
MEN (aufgerufene Methode existiert nicht)	1610	7 %
KIE (Klasse, Interface oder Enum erwartet)	1540	7 %
UAN (ungültige Anweisung)	1278	6 %
MUP (Methodenaufruf mit ungültigen Parametern)	1112	5 %

Tabelle 3.12.: Verteilung der häufigsten Syntaxfehler im Programmcode von Roboter-Klasssen nach Typ

Bei Betrachtung von Tabelle 3.11 fällt auf, dass die Häufigkeit von Fehlern offenbar stark von der bearbeiteten Aufgabe abhängt. Während beim Szenario „LabyrinthRandom“ nur etwa jede zehnte Lernerlösung bezüglich der Syntax des Programmcodes fehlerhaft war, war es bei der „Kaninchenjagd“ mehr als jede zweite.

Die mit Abstand häufigsten Fehler waren fehlende öffnende und schließende Klammern oder ein fehlendes Semikolon (siehe Tabelle 3.12). Diese Art von Fehlern tritt auch bei der manuellen Erstellung von Java-Code sehr häufig auf.

Auch zu den bei jedem Aufruf der Arena erzeugten Animationen wurden Statistiken erfasst (siehe Tabelle 3.13). Es wurden insgesamt über 16.000 Animationen von etwa 270 verschiedenen Benutzern generiert und betrachtet (dritte Feedback-Schleife). Die technisch bedingte, maximale Anzahl an Animationsschritten betrug dabei 1000. Danach 1000 wurde die Animation abgebrochen.

Wie in Tabelle 3.13 zu sehen ist, hängt die Anzahl an tatsächlich ausgeführten Animationsschritten stark vom bearbeiteten Szenario ab. Auffällig ist auch, dass die Anzahl

	RE	**HO**	**HOR**	**LAB**	**KAN**	**gesamt**
Anzahl Animationen	2343	2904	4568	2218	4137	16.170
Anzahl verschiedener Benutzer	266	257	238	236	179	273
Anzahl verschiedener Roboter-Klassen	1863	2407	3330	1502	2176	11.187
Anteil erfolgreiche Ausführungen	30 %	34 %	23 %	38 %	35 %	31 %
Ø Anzahl an Animationsschritten	86	331	423	484	72	276
Ø Anzahl an Animationsschritten (nur erfolgreiche Ausführungen)	37	382	420	201	51	216

Tabelle 3.13.: Statistiken zu den erzeugten und angesehenen Animationen der automatisch generierten Roboter-Klassen im Sommersemester 2014 (RE = Rechteck, HO = Hoover, HOR = HooverRandom, LAB = LabyrinthRandom, KAN = Kaninchenjagd)

	RE	**HO**	**HOR**	**LAB**	**KAN**	**gesamt**
Anzahl Einreichungen	256	243	209	221	144	1073
Anteil bestandener Einreichungen	98,0 %	98,8 %	89,5 %	98,2 %	88,2 %	95,3 %
Ø Anzahl Aktivitäten	2,5	3,0	3,8	2,3	3,4	2,9
Ø Anzahl Knoten	17,2	21,5	39,3	12,9	34,5	23,2
Ø Anzahl Kanten	15,0	17,9	30,9	9,9	18,5	18,2

Tabelle 3.14.: Anzahl der Einreichungen sowie der Anteil als bestanden gewerteter Einreichungen der fünf bearbeiteten Szenarien im Sommersemester 2014

der ausgeführten Animationsschritte bei erfolgreichen Ausführungen je nach Szenario teilweise deutlich geringer ist als bei nicht erfolgreichen Ausführungen. Beim Szenario „HooverRandom“ verhält es sich invers. Auch weist dieses den geringsten Anteil an erfolgreichen Ausführungen auf.

Für jedes zu bearbeitende Szenario mussten die Studierenden eine Roboter-Klasse (und damit ein Diagramm) als finale Lösung einreichen. Pro Studierendem gibt es also maximal eine Einreichung je Szenario. Dieser Roboter wurde nach der Einreichung erneut im entsprechenden Szenario ausgeführt. Wurde die Aufgabe des Szenarios in diesem Durchlauf erfüllt, so galt dieses Szenario als bestanden. Es wurden mehr als 1000 verschiedene Roboter-Klassen von mehr als 260 verschiedenen Studierenden eingereicht.

Tabelle 3.14 zeigt die Anzahl der Einreichungen, den Anteil als bestanden gewerteter Einreichungen, sowie weitere Statistiken über die entsprechenden Aktivitätsdiagramme. Die Tabellen 3.15 und 3.16 zeigen die durchschnittliche Benutzung der Knotentypen und der vorgegebenen aktorischen und sensorischen Befehle.

In Tabelle 3.14 ist anhand der Bestehensquoten zuerkennen, dass den Studierenden die Lösung der Aufgaben „HooverRandom“ (HOR) und „Kaninchenjagd“ (KAN) wesentlich schwerer fiel als die der anderen Szenarien.

	RE	**HO**	**HOR**	**LAB**	**KAN**	**gesamt**
Aktionsknoten	14/100 %	17/100 %	30/100 %	9/100 %	25/100 %	18/100 %
Verzweigungsknoten	0,5/23 %	0,8/34 %	1,6/46 %	1,0/35 %	1,4/36 %	1,0/34 %
Verbindungsknoten	0,3/14 %	0,5/23 %	1,3/40 %	0,8/28 %	1,1/27 %	0,7/25 %
If-Then-Knoten	0,2/9 %	1,2/55 %	3,0/69 %	1,1/71 %	5,4/91 %	1,6/52 %
For-Do-Knoten	0,9/38 %	0,1/5 %	0,1/8 %	0,0/0 %	0,0/4 %	0,3/13 %
While-Do-Knoten	1,4/53 %	1,9/84 %	2,5/89 %	0,8/76 %	1,4/87 %	1,6/75 %
Do-While-Knoten	0,0/1 %	0,2/12 %	0,2/14 %	0,0/0 %	0,0/1 %	0,1/6 %

Tabelle 3.15.: Nutzung der Knoten nach Typ im Sommersemester 2014: Der erste Wert gibt den Mittelwert der in den Diagrammen enthaltenen Knoten des entsprechenden Typs an, der zweite Wert gibt den Anteil an Diagrammen an, die mindestens einem Knoten entsprechenden Typs beinhalten.

Auch fällt auf, dass die durchschnittliche Anzahl an vorkommenden Knoten wesentlich höher ist als im Vorjahr. Gerade für das stochastische Szenario „HooverRandom“ benötigten die Studierenden gegenüber der einfacheren Variante „Hoover“ (HO) deutlich mehr Knoten. Trotz der Erweiterung des Szenarios „Labyrinth“ aus dem Vorjahr um ein stochastisches Element für das Szenario „LabyrinthRandom“ (LAB) blieb hier die durchschnittliche Anzahl verwendeter Knoten etwa gleich. Dies ist nicht verwunderlich, da nur die Startposition des Roboters im Labyrinth randomisiert wurde, was für einen Lösungsalgorithmus keine erhöhte Komplexität bedeutet.

Dass mit Interpretationen dieser Zahlen vorsichtig umgegangen werden muss, zeigt die Tatsache, dass für das gegenüber dem Vorjahr unverändert gebliebene Szenario „Rechteck“ mit einem Durchschnitt von 17,2 deutlich Knoten verwendeten wurden als zuvor mit 12,8 Knoten. Die Ursache unterschiedlicher Nutzungsstatistiken muss also nicht die Veränderung eines Szenarios sein. Auch andere Faktoren können eine Rolle spielen, beispielsweise eine verschiedene Vorbereitung auf die Aufgaben während der Präsenzlehre oder schlicht die verschiedene Leistungsfähigkeit einer Kohorte.

Tabelle 3.15 zeigt, dass sich auch eine Veränderung bei der Nutzung von Knotentypen bei unverändertem Szenario ergeben kann, zeigt . In mehr als 50 % der Aktivitätsdiagramme für das Szenario „Rechteck“ waren hier strukturierte While-Do-Knoten enthalten, im Vorjahr dagegen traten sie nur bei 4 % der Diagramme auf. Dies war nicht zu erwarten, da im Sommersemester 2014 die Möglichkeit bestand, Schleifen auch durch die Verwendung von Verzweigungsknoten abzubilden und die Verwendung von While-Do-Knoten somit nicht nötig war.

	RE	**HO**	**HOR**	**LAB**	**KAN**	**gesamt**
moveForward()	2,5/99 %	3,7/98 %	10,0/99 %	2,5/100 %	5,1/100 %	4,6/99 %
rotateRight()	1,9/65 %	2,7/95 %	6,2/99 %	1,8/95 %	1,9/99 %	2,9/88 %
rotateLeft()	1,0/39 %	2,3/94 %	6,0/98 %	1,5/91 %	1,5/96 %	2,4/80 %
moveRandom()	0,0/0 %	0,0/0 %	0,0/3 %	0,1/5 %	0,2/16 %	0,0/3 %
markCurrentNode()	2,5/99 %	3,5/99 %	5,3/99 %	0,0/2 %	0,1/9 %	2,4/71 %
canGoForward()	0,1/3 %	1,9/86 %	2,8/93 %	1,0/70 %	0,8/77 %	1,3/62 %
canGoLeft()	0,0/0 %	0,1/14 %	0,6/33 %	0,6/37 %	0,3/47 %	0,3/22 %
canGoRight()	0,0/1 %	0,2/22 %	0,6/41 %	0,8/60 %	0,3/43 %	0,4/30 %
isMarked()	0,0/2 %	0,4/25 %	0,5/28 %	0,9/78 %	0,0/0 %	0,4/30 %
thereIsARabbit()					0,5/83 %	
getForwardDistanceToRabbit()					1,1/96 %	
getBackwardDistanceToRabbit()					0,8/79 %	
getLeftDistanceToRabbit()					0,9/84 %	
getRightDistanceToRabbit()					0,9/88 %	
shoot()					0,9/99 %	

Tabelle 3.16.: Nutzung der vorgegebenen aktorischen und sensorischen Befehle im Sommersemester 2014: Der erste Wert gibt den Mittelwert der in den Diagrammen enthaltenen Befehle , der zweite Wert gibt den Anteil an Diagrammen an, in denen der Befehl mindestens einmal vorkommt (die letzten sechs Befehle standen nur im Szenario „Kaninchenjagd“ (KAN) zur Verfügung)

Dass diese Möglichkeit genutzt wurde, ist ebenfalls aus Tabelle 3.15 ablesbar: Für jedes Szenario ist die Anzahl an verwendeten Verzweigungsknoten höher als die der Verbindungsknoten. Da bei einer Modellierung von Verzweigungen ohne Schleifen, die im Programmcode durch *If-Then*-Strukturen abgebildet werden, bei vorausgesetzter syntaktischer Korrektheit immer genau ein Verzweigungs- und genau ein Verbindungsknoten vorkommt (siehe Abbildung 3.4 auf Seite 82), kann gefolgert werden, dass wegen der hier vorkommenden höheren Nutzung von Verzweigungsknoten gegenüber Verbindungsknoten tatsächliche Schleifen modelliert wurden.

In Tabelle 3.16 zeigt sich, dass auch hier das unveränderte Szenario „Rechteck“ zu veränderter Modellierung führte. Ließ im Vorjahr ein Großteil der Studierenden ihren Roboter das Rechteck noch durch Rotationen nach links gegen den Uhrzeigersinn laufen, verhielt es sich im Sommersemester 2014 invers.
Außerdem fällt auf, dass fast jeder zehnte Studierende bei der „Kaninchenjagd“ den Roboter Felder markieren ließ, was bei der Lösung dieser Aufgabe nicht zielführend ist. Hier könnte nun durch Betrachtung der entsprechenden Aktivitätsdiagramme durch Lehrpersonal untersucht werden, wieso diese mutmaßlichen Fehler aufgetreten sind, so dass im Folgejahr in der Vorbereitung darauf eingegangen werden kann.

Wenige Tage nach Bearbeitung der Aufgaben wurden die Studierenden gebeten, einen Online-Fragebogen auszufüllen, in welchem auch konkrete Fragen zu CaseTrain-WARP

Aspekt	Ø Note
Konzept des Tools, unabhängig von der Implementierung	1,91
Umsetzung des Tools, Gesamtnote	3,24
Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)	3,59
Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)	3,87
Aspekt Inhalt der Übungsaufgaben	2,66

Tabelle 3.17.: Umfrageergebnisse zu CaseTrain-WARP im Sommersemester 2014. Die verschiedenen Aspekte wurden mit Schulnoten (1-6) bewertet (n=92).

Aspekt	Häufigkeit
Visualisierung (Animation)	16
Idee/Konzept	10
interessante Aufgaben	7
Spaß/Motivation	5
erkennbarer Zusammenhang von Code & Diagramm	3
Fehlermeldungen für Diagram & Programmcode	3
Interaktivität	3
Einsteigerfreundlichkeit	2

Tabelle 3.18.: Die am häufigsten genannten positiven Aspekte von CaseTrain-WARP. Einmalig genannt wurden noch: optische Ästhetik, allgemeine Verfügbarkeit, Innovation, Umsetzung, Effizienz und einfache Bedienung. (insgesamt 39 Antworten)

gestellt wurden (siehe Anhang Seite 182). Die Studierenden sollten bestimmte Aspekte der Anwendung mit Schulnoten (1-6) bewerten. Tabelle 3.17 zeigt die Ergebnisse dieser Umfrage. Die Frage, ob WARP in Zukunft auch weiterhin im Übungsbetrieb eingesetzt werden sollte, beantworteten 78 % der Studierenden mit „Ja“.

Die Studierenden konnten in der elektronischen Umfrage mit einem Freitext angeben, was ihnen gut und was ihnen nicht gefallen hat. Die Fragen wurden von 39 (positive Aspekte) bzw. 55 Studierenden (negative Aspekte) beantwortet. Die genannten Aspekte wurden zusammengefasst und solche, die mehr als einmal genannt wurden, sind in den Tabellen 3.18 bzw. 3.19 dargestellt (vollständige Antworten siehe Anhang ab Seite 182).

3.5.6. Diskussion

Aufgrund der durchgeführten Experimente lässt sich sagen, dass CaseTrain-WARP effektiv bedienbar ist. Vor allem die Anzahl der Studierenden, die ausführbare Programme erzeugen konnten und die hohen Erfolgsquoten belegen die Effektivität der Anwendung.

Aspekt	Häufigkeit
Technik (Bugs, Robustheit, Verfügbarkeit, Ladezeiten)	27
Aufgaben zu schwer/unfair/unpräzise/umfangreich	20
Bedienung (kompliziert bzw. fehlende Features)	9
Editor: keine copy&paste Funktion	9
Platzbedarf / Aufteilung GUI	6
Editor: verschieben in Block-Container nicht möglich	5
fehlender „Debug“-Modus	3
Editor: fehlende Hilfe-Funktion	2
Fehlermeldungen für Diagram & Programmcode	2
Datenverlust bei Browser-Navigation	2

Tabelle 3.19.: Die am häufigsten genannten negativen Aspekte von CaseTrain-WARP. Einmalig genannt wurden noch: Effizienz, Bewertung der Lösungen, Verwendung von Flash, fehlende Einarbeitung und zu wenige vorgegebene Befehle. (insgesamt 55 Antworten)

3.5.6.1. Experiment im Sommersemester 2013

Die Studierenden im Sommersemester 2013 empfanden das Konzept grundsätzlich als gut, waren aber mit der Umsetzung und der Bedienbarkeit teilweise nicht zufrieden. Vor allem der Aspekt „Technik (Verfügbarkeit, Robustheit, Effizienz)“ wurde mit einer durchschnittlichen Note von 3,81 negativ bewertet. Eine mögliche Erklärung ist, dass die Webanwendung in der damaligen Implementierung unmittelbar implizit nach jeder Änderung eines Aktivitätsdiagramms (unter Beibehaltung syntaktischer Korrektheit) den zugehörigen Programmcode generierte und diesen, sofern er auch syntaktisch korrekt war, ebenfalls unmittelbar kompilierte und eine entsprechend RoSE-Animation generierte. Die dadurch erzeugte hohe Anzahl an Server-Anfragen mit teilweise sehr hoher Prozessor-Auslastung (für die Generierung der Animationen) sorgte in Verbindung mit vielen gleichzeitigen Benutzern vor allem am Ende der erlaubten Bearbeitungszeit dafür, dass die Webanwendung zeitweise nur sehr unzuverlässig erreichbar war und Studierende die Aufgabe somit nicht bearbeiten konnten. In der Tat wurden Aufgaben, die in der gleichen Woche zu bearbeiten waren, von etwa 290 Studierenden bearbeitet, so dass zu befürchten ist, dass zwischen 15 % (nur 244 Einreichungen für „Rechteck“) und 28 % (nur 206 Einreichungen für „Labyrinth“) der Studierenden die Anwendung nicht verwenden bzw. kein lauffähiges Programm erstellen konnten.

Auch die erzwungene Verwendung der als unsicher geltenden Java-Applets ist ein möglicher Grund für schlechte Bewertungen.

Die Aufgaben wurden inhaltlich mit einer durchschnittlichen Note von 2,19 größtenteils positiv eingeschätzt, auch mit der Fairness der Bewertung der Aufgaben waren die Lernenden mit einer Note von 2,33 weitgehend zufrieden. Mit einer durchschnittlichen Note von 2,81 wurden die Erklärungen und das generierte Feedback etwas schlechter bewertet.

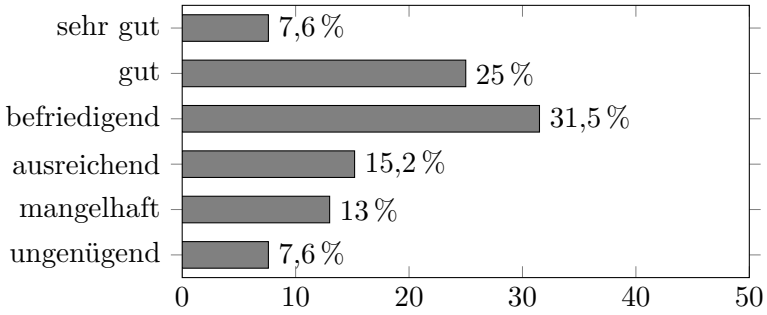


Abbildung 3.11.: Verteilung der Gesamtnote für CaseTrain-WARP im Sommersemester 2014 ($n = 92$; $mw = 3{,}24$; $\sigma = 1{,}37$)

3.5.6.2. Experiment im Sommersemester 2014

Das Konzept von CaseTrain-WARP wurde von den Studierenden im Sommersemester 2014 mit einer Note von 1,91 noch etwas besser bewertet als im Vorjahr. Die am häufigsten als positiv empfundenen Aspekte waren die Visualisierung des Verhaltens des Roboters per Animation, die Idee an sich, die Interessantheit der Aufgaben und dass die Bearbeitung selbst Spaß bereitete (siehe Tabelle 3.18).

Jedoch wurde die Umsetzung der Anwendung (3,24), die Bedienbarkeit (3,59) und die technische Umsetzung (3,87) eher kritisch bewertet. Diese Aspekte wurden von den Studierenden allerdings sehr unterschiedlich wahrgenommen. Abbildung 3.11 zeigt die Verteilung der Noten für die Umsetzung der Anwendung (Gesamtnote). Es gab also sowohl Studierende, die die Anwendung mit „sehr gut“ oder „gut“ (32,6 %), als auch Studierende, die sie mit „mangelhaft“ oder „ungenügend“ (20,7 %) bewerteten.

Da das Konzept gut bewertet wurde, liegen die Ursachen für die schlechten Bewertungen wahrscheinlich in den anderen negativ bewerteten Aspekten, also Technik und Bedienbarkeit. Dies legen die Benotungen der entsprechenden Aspekte (Tabelle 3.17) und die in den Freitext-Antworten genannten, als negativ empfundenen Aspekte (Tabelle 3.19) nahe.

Die Technik wurde auch im vorigen Semester schon mit einer Note von 3,57 eher schlecht bewertet. Die für das Experiment im Sommersemester 2013 vermutete hauptsächliche Ursache, nämlich die Serverausfälle aufgrund zu hoher Serverlast, wurde durch ein verändertes technisches Konzept bei der Feedback-Generierung behoben (explizite statt implizite Server-Anfragen). Auch wurde auf die Verwendung eines Java-Applets zugunsten von Standard-Webtechnologien verzichtet.
Dennoch wurde der Aspekt Technik im Sommersemester 2014 schlechter bewertet. Eine mögliche Ursache ist, dass die Anwendung in den ersten 3-4 Tagen des Produktiveinsatzes fehlerbehaftet war, konkret funktionierte die Umwandlung der Aktivitätsdiagramme in Programmcode nicht immer korrekt. Dass diese Problematik erst im Experiment im Jahr 2014, nicht aber im Vorjahr auftrat, ist einer Erweiterung der erlaubten Syntax der Aktivitätsdiagramme geschuldet, die ab 2014 auch Zyklen zur Abbildung von Schleifen

erlaubte. Die Umwandlung dieser Zyklen in Programmcode erforderte eine Erweiterung der entsprechenden Implementierung, welche offenbar nicht ausreichend getestet wurde, so dass Programmfehler, die bei bestimmten Diagramm-Konstrukten auftraten, erst in der Produktivphase der Anwendung erkannt und behoben werden konnten. Dies sorgte bei vielen Studierenden für Unmut. Aufgrund dieser Fehler wurden Aktivitätsdiagramme nicht oder gar fehlerhaft in Programmcode übersetzt, so dass die Studierenden in der Animation nicht das tatsächlich im Aktivitätsdiagramm modellierte Verhalten sehen konnten. Dass eine solche gravierende Fehlfunktion zu einer schlechteren Bewertung führt, ist sehr wahrscheinlich.

Die Bedienbarkeit wurde mit einer Note von 3,59 um fast eine Notenstufe schlechter als im Vorjahr bewertet, obwohl das Bedienkonzept, und vor allem der Aktivitätsdiagramm-Editor selbst nicht verändert wurde, in welchem die hauptsächliche Benutzerinteraktion stattfinden.
Es wird vermutet, dass die Ursache hier in einer höheren Komplexität der erarbeiteten Aktivitätsdiagramme im Sommersemester 2014 liegt. Die Einführung der stochastischen Elemente in den Szenarien „Hoover“ und „Labyrinth“ sowie des Mehrspieler-Szenarios „Kaninchenjagd“ erhöhte die Komplexität der entsprechenden Aufgaben und führte zu größeren eingereichten Diagrammen: Während im Vorjahr die Diagramme mit durchschnittlich etwa 15 Knoten und 13 Kanten (siehe Tabelle 3.4) noch relativ klein waren, wuchsen sie im Sommersemester 2014 auf durchschnittlich 23 Knoten und 15 Kanten (siehe Tabelle 3.14) signifikant[7] an. Bei den Szenarien „HooverRandom“ und „Kaninchenjagd“ waren es durchschnittlich sogar 39 und 35 Knoten. Zusätzlich modellierten die Studierenden auch relativ wenige Aktivitäten, so dass in jeder Aktivität sehr viele Knoten enthalten waren. Bei den Szenarien „HooverRandom“ und „Kaninchenjagd“ waren es durchschnittlich mehr als 10 Knoten pro Aktivität.
Problematisch an größeren Aktivitätsdiagrammen ist nun der Editor, der für kleine Diagramme bzw. Aktivitäten mit wenigen Knoten konzipiert wurde. Vor allem das stark eingeschränkte Platzangebot ist hier ein Problem: Die Breite der Zeichenfläche ist fest vorgegeben[8], so dass die Graphen innerhalb der Aktivitäten „von oben nach unten“ gezeichnet werden müssen, was der ursprünglichen Vorstellung der Benutzer, also deren internem optischen Modell, widersprechen könnte.
Auch dann, wenn Anwender Änderungen am Layout besonders komplexer Diagrammen vornehmen wollen, kann dies sehr aufwändig sein, da große Teile des Graphen verschoben werden müssen. Dies kann dadurch erschwert werden, dass bei einer gewünschten Verschiebung von Knoten in den aktuell nicht sichtbaren Bereich im Editor derzeit nicht automatisch weiter gescrollt wird, so dass das Verschieben in mehreren Schritten vorgenommen werden muss.
Es ist also naheliegend, dass die erhöhte inhaltliche Komplexität der Aufgaben zu einer schlechteren Bewertung der Bedienbarkeit führte, da sich die Einschränkungen der Ge-

[7] Mann-Whitney-U-Test mit einem Signifikanzniveau von unter 0,1 %
[8] Dies war konzeptionell bei der Entwicklung dieses (nicht nur in WARP) eingesetzten Editors so vorgesehen, um erstellte Diagramme problemlos auf DINA4 Papier ausdrucken zu können

brauchstauglichkeit des Editors bei komplexen Diagrammen weitaus stärker bemerkbar machen als bei einfachen Diagrammen.

Die Einführung des Mehrspieler-Szenarios im Sommersemester 2014 hat entgegen der Erwartungen nicht zu einer Verbesserung der durchschnittlichen Note für den Aspekt „Inhalt der Übungsaufgaben“ geführt: die Bewertung fiel mit einer durchschnittlichen Note von 2,66 sogar schlechter aus als im Vorjahr (2,19). Eine mögliche Erklärung ist, dass diese Mehrspieler-Aufgabe, ebenso wie die durch stochastische Elemente erweiterte Aufgabe „HooverRandom“ von den Studierenden als zu schwer bzw. zu umfangreich wahrgenommen wurde. Diese These wird von den Freitext-Antworten aus dem Fragebogen (siehe Tabelle 3.19) unterstützt.

3.6. Zusammenfassung und Ausblick

Es wurde ein Konzept vogestellt, bei dem Lernende beim freien Zeichnen von UML Aktivitätsdiagrammen ein mehrstufiges Feedback erhalten. Dieses Konzept wurde in Form der Webanwendung CaseTrain-WARP implementiert und in zwei aufeinanderfolgenden Vorlesungen im Übungsbetrieb mit etwa 500 Studierenden eingesetzt und evaluiert. Das Konzept der Webanwendung, und somit das in Abschnitt 3.2 vorgestellte Konzept, wurde von den Studierenden für gut befunden. Dies wird neben der Bewertung des Konzepts mit durchschnittlichen Noten von 2,28 bzw. 1,91 vor allem von der Tatsache belegt, dass trotz der Probleme bei der Bedienung und der technischen Umsetzung 78 % der Studierenden im Sommersemester 2014 der Meinung sind, dass WARP in Zukunft auch weiterhin im Übungsbetrieb eingesetzt werden sollte. Zudem sprachen sich im Sommersemester 2013 94 % der Studierenden allgemein für den Einsatz elektronischer Tools im Übungsbetrieb aus. Die Fallstudie ist daher als Erfolg zu werten. Dennoch gilt es, die Anwendung auf verschiedenen Ebenen zu verbessern.

3.6.1. Technische Verbesserungen

Um eine breitere Akzeptanz bei den Studierenden zu erreichen, müssen die technischen Probleme in Zukunft deutlich verringert werden. Dies umfasst Programmfehler ebenso wie die Erreichbarkeit der Webanwendung. Zudem muss der Aktivitätsdiagramm-Editor genauer auf seine Gebrauchstauglichkeit hin überprüft und entsprechend angepasst bzw. neu implementiert werden. Die wichtigsten vorzunehmenden Änderungen dabei sind:

- **Variables Platzangebot**: Es sollte für den Benutzer möglich sein, die Zeichenfläche auch in der Breite zu vergrößern. Es müssen dann auch entsprechende Möglichkeiten implementiert werden, den sichtbaren Teil der Zeichenfläche zu verschieben und ggf. zu zoomen. Zwar würden sich so erstellte Diagramme nicht mehr unkompliziert auf DINA4 Papier drucken lassen, doch erscheint dieser Nachteil für den Einsatz in WARP zweitrangig.

- **Verschieben von Elementen in Knoten-Container**: Aktuell ist es nicht möglich, Teile des Graphen (z. B. einen Knoten) in einen Knoten-Container (z. B. in den Then-Block eines strukturierten If-Then-Knotens) des gleichen Graphen zu verschieben. Dies ist aber dann ein Problem, wenn ein Subgraph wie gewünscht modelliert wird, dann allerdings noch von einer Verzweigungs- oder Schleifen-Struktur umgeben werden soll. Wenn der Benutzer dazu keinen Verzweigungsknoten, sondern einen strukturierten Knoten benutzen will, muss der Subgraph derzeit gelöscht und im strukturierten Knoten neu erstellt werden, was eine erhebliche Einschränkung der Gebrauchstauglichkeit bedeutet. Diese Möglichkeit sollte also implementiert werden.
- **Copy&Paste- bzw. Cut&Paste-Funktion**: Dieses Funktion hängt mit der zuletzt genannten zusammen, würde allerdings noch etwas mehr Möglichkeiten bieten. Derzeit existiert keine Möglichkeit, Subgraphen zu kopieren (bzw. auszuschneiden) und an anderer Stelle (beispielsweise auch in einer anderen Aktivität wieder einzufügen). Die entsprechende Funktionalität sollte implementiert werden.
- **Alternative Eingabe von Name und Ein-/Rückgabe-Parametern einer Aktivität**: Momentan werden Name und Ein-/Rückgabe-Parametern einer Aktivität mittels der Eingabe einer Methodensignatur in Java-Syntax definiert. Die Signatur wird vom Editor geparst und die entsprechenden Informationen werden ausgelesen, wobei es (abgesehen vom generierten Programmcode) kein weiteres Feedback, insbesondere keine Fehlerbehandlung gibt. Da Studierenden in Anfängervorlesungen die Java-Syntax oftmals nicht ausreichend vertraut ist, sollte hier eine alternative, wesentlich einfachere Eingabemöglichkeit implementiert werden. Zunächst sollte entschieden werden, ob es überhaupt nötig ist, dass die Wahl der Modifikatoren den Benutzern überlassen bleibt. Da das Lernziel hauptsächlich in der Erstellung von Aktivitätsdiagrammen und nicht in der Erstellung von Programmcode liegt, könnten diese Modifikatoren fest vorgegeben sein, so dass beispielsweise alle erstellten Methoden private, nicht-statische Methoden sind. Dies würde, wie erwünscht, eine Vereinfachung darstellen.
  Zusätzlich sollte für die Ein- und Rückgabeparameter neben Texteingabefeldern für die Namen eine Vorauswahl von Datentypen (z. B. per Drop-Down-Liste) angeboten werden.
- **leichter erkennbare Schaltflächen**: Die Schaltflächen zum Hinzufügen und Entfernen von Blöcken bei strukturierten If-Then-Knoten (z. B. Else-Block) sind sehr klein und nicht selbstbeschreibend genug. Hier sollte durch besser erkennbare Schaltflächen oder ein anderes Bedienkonzept nachgebessert werden.
- **Einbinden von vorgegebenen Befehlen**: Die Eingabe der vorgegebenen aktorischen und sensorischen Befehle (siehe Tabellen 3.3 und 3.8) wird aktuell nur durch die Funktion des Auto-Vervollständigens unterstützt. Die Analyse der aufgetretenen syntaktischen Fehler im generierten Quellcode (siehe Tabelle 3.12) ergab, dass sich ein großer Teil (etwa 50 %) der meist durch Schreib- oder Tippfehler verursachten Fehler des Typs MEN (Methode existiert nicht) auf die vorgegebenen Befehle

bezog. Beispielsweise wurde statt `rotateRight()` fälschlicherweise `rotateright()` oder `turnRight()` eingegeben. Diesen Fehlern kann vorgebeugt werden, indem für den Editor eine Funktion implementiert wird, mit der Anwender vorgegebene Befehle unkompliziert und korrekt in ihre Aktivitäten einbinden können, beispielsweise wie in Alice per Drag&Drop [Cooper et al., 2000].
Die Funktionalität kann auch so angepasst werden, dass von Anwendern erstellte Aktivitäten dynamisch zur Liste der vorgegebenen Befehle hinzugefügt werden, so dass auch hier Fehler beim Methodenaufruf verhindert werden können.

3.6.2. Organisatorische Verbesserungen

Zwar gab es im Sommersemester eine schriftliche Kurzanleitung zur Bedienung von WARP (Anhang ab Seite 195), allerdings mussten sich die Studierenden einige Bedienkonzepte des Aktivitätsdiagramm-Editors selbst erarbeiten, da diese nicht ausreichend selbstbeschreibend waren. Es erscheint sinnvoll, zusätzlich zu den bereits angebotenen Screencasts, in welchen die Erstellung nur eines Aktivitätsdiagramms mit wenigen Elementen bis zur Einreichung gezeigt wurde, weitere sehr kurze Screencasts zur Benutzung verschiedener Konzepte des Editors zur Verfügung zu stellen.

3.6.3. Didaktische Verbesserungen

Um eine Verringerung der Komplexität der Aufgaben bei gleichbleibender Interessantheit zu erreichen, sollten die in den Abschnitt 3.3.1 und 3.3.2 vorgestellten Ansätze umgesetzt werden (Korrekturen in vollständigen Aktivitätsdiagrammen und Aktivitätsdiagramm-Vorlagen). So kann noch mehr nach dem Ansatz der steigenden Schwierigkeit oder beispielsweise nach dem *Cognitive Apprenticeship Modell* vorgegangen werden, womit den Studierenden der Einstieg erleichtert werden kann.

3.6.4. Verbesserung durch regelbasierte Überprüfungen

Das beschriebene Verfahren sollte um regelbasierte Überprüfungen der Aktivitätsdiagramme erweitert werden. Momentan werden alle Diagramme als korrekte Lösungen akzeptiert, die zum gewünschten Ergebnis führen. Doch ist das korrekte Ergebnis der Ausführung eines übersetzten Aktivitätsdiagramms nicht das alleinige Kriterium für dessen Qualität.
Beispielsweise lässt sich das Konzept der in Abschnitt 3.2 beschriebenen Code Smells auch auf Aktivitätsdiagramme übertragen. Denn auch wenn in WARP ausführbare Aktivitätsdiagramme erstellt werden, sollte der Fokus bei der Modellierung doch auf einer übersichtlichen und somit leicht verständlichen Darstellung der modellierten Routinen liegen. Einige Code Smells lassen sich dazu direkt auf Aktivitätsdiagramme abbilden, beispielsweise duplizierter Code oder eine zu lange Parameterliste.
Es lassen sich allerdings auch eigene „Diagram Smells“ definieren, die letztlich als Heuristik dienen, um festzustellen, ob ein Aktivitätsdiagramm übersichtlich gestaltet ist oder

nicht. Als Kriterien können hier beispielsweise die Anzahl von Knoten je Aktivität sein, die nicht zu hoch sein sollte. Auch die Anzahl an Zeichen pro Aktion sollte möglichst gering sein. Gleiches gilt für die Anzahl an Schleifen und Verzweigungen in Aktivitäten. Ebenfalls feststellen ließe sich, ob sich die Knoten in den Aktivitäten gleichmäßig auf der Zeichenfläche verteilen und nicht zu eng beieinander liegen. Auch sich kreuzende Verbindungslinien könnten als solcher Diagram Smell definiert werden.
Entsprechende regelbasierten Überprüfungen könnten vor, nach, oder zusammen mit der ersten Feedback-Schleife (syntaktischen Überprüfung der Diagramms) stattfinden.

Durch solche Regeln und entsprechende Rückmeldungen könnten Studierende auch dahingehend motiviert werden, nach dem in Abschnitt 3.1 beschriebenen Top-Down-Prozessmodell zu Erstellung von Klassendiagrammen vorzugehen.

4. Anfragen an relationale Datenbanken

Ein wichtiges Thema im Studium der Informatik und verwandten Studiengängen sind relationale Datenbanken. Neben dem effizienten Design der Struktur von Datenbanken, bei dem beispielsweise Normalformen eingehalten werden sollten, stellen Anfragen an diese einen zentralen Aspekt im Umgang mit Datenbanken dar. Die *Structured Query Language* (SQL) ist eine standardisierte Datenbanksprache, mit der relationale Datenbanken, definiert, bearbeitet und abgefragt werden können. Viele weit verbreitete Datenbank Management Systeme (DBMS), wie beispielsweise MySQL, PostgreSQL, Miscrosoft SQL Server, IBM DB oder Oracle Database erfüllen den SQL Standard, wobei deren Abfragesprachen sogenannte SQL-Dialekte bilden. Im Informatik-Studium sollten relationale Datenbanken nicht nur theoretisch behandelt, sondern auch praktisch eingeübt werden. Es ist daher wichtig, ausreichend viele Übungsmöglichkeiten anzubieten.

In diesem Kapitel wird ein Verfahren vorgestellt, bei dem Lernende für eingegebene SQL Anfragen entsprechendes Feedback und eine Bewertung erhalten. Das Verfahren wurde implementiert und im Übungsbetrieb einer Vorlesung über zwei Semester eingesetzt und evaluiert.

4.1. Beschreibung des Aufgabentyps

Beim hier vorgestellten Aufgabentyp steht nicht die Erstellung der Datenbanken, sondern Anfragen an diese im Mittelpunkt. Den Lernenden wird eine Domäne mit einer zugehörigen Datenbank präsentiert, die mit entsprechenden Daten gefüllt ist. Es werden nun Aufgaben gestellt, die die Lernenden durch einen Aufgabentext jeweils dazu auffordern, eine solche Anfrage an die Datenbank zu stellen, die zu einem gewünschten Ergebnis führt. Dabei beschränken sich die geforderten Anfragen auf solche der Typen: `SELECT`, `INSERT`, `UPDATE` und `DELETE`. Die Lernziele sind:

- allgemeiner Umgang mit einer vordefinierten relationalen Datenbank
- Eingabe syntaktisch korrekter SQL Anfragen
- Eingabe semantisch korrekter SQL Anfragen

Es handelt sich bei diesem Aufgabentyp um eine offene, stark strukturierte Aufgabe: Lernende müssen hier eine Lösung konstruieren, nicht selektieren. Zudem existiert keine eindeutige, korrekte Lösung der Aufgabe, da es oft mehrere Möglichkeiten gibt, eine syntaktisch und semantisch korrekte SQL Anfrage entsprechend der Aufgabenstellung zu erstellen. Die starke Strukturierung der Aufgabe ergibt sich aus der Syntax der SQL. Tabelle 4.1 zeigt einige Beispiele zu Aufgaben aus einem Szenario in der Domäne einer Fußball-Weltmeisterschaft:

Aufgabentext	Musterlösungs-Anfragen

Wie viele cm ist Thomas Müller größer als Javier Mascherano? Bennen Sie die Ergebnisspalte „difference“.	`SELECT (SELECT size FROM person WHERE name='Thomas Müller') - (SELECT size FROM person WHERE name='Javier Mascherano') AS difference`
	
	`SELECT MAX(size)-MIN(size) AS difference FROM person WHERE name='Thomas Müller' OR name='Javier Mascherano'`

Wie viele Linksfüßer (footer = 'links') hatte Deutschland dabei? Benennen Sie die Ergebnisspalte „number“.	`SELECT COUNT(*) AS number FROM player, association WHERE player.association = association.id AND association.team_name = 'Deutschland' AND player.footer = 'links'`
	
	`SELECT COUNT(person_id) as number FROM player JOIN association ON player.association = association.id WHERE footer = 'links' AND team_name='Deutschland'`

Wolfgang Stark (id = 795) wird beim Eröffnungsspiel (match_id = 1) als vierter Offizieller (role = 'assistant') hinzugezogen. Fügen Sie ihn in die Tabelle „match_has_refs“ ein.	`INSERT INTO match_has_refs (match_id, referee_id, role) VALUES (1,795,'assistant')`
	
	`INSERT INTO match_has_refs ('1','795','assistant')`

Der Spieler Thomas Enevoldsen ist krank und reist ab. Löschen sie ihn aus der Personentabelle.	`DELETE FROM person WHERE name = 'Thomas Enevoldsen'`

Tabelle 4.1.: Beispiele von Aufgaben aus einem Szenario zur Organisation einer Fußball-Weltmeisterschaft

4.2. Verwandte Arbeiten

In der Literatur existieren bereits einige Ansätze für Trainingssysteme für SQL-Anfragen. Zugehörige Studien zeigen den großen Nutzen von SQL-Trainingssystemen für Studierende: 84 von 94 Studierenden würden das Trainingssystem SQLT-Web [Mitrovic, 2003] anderen Studierenden weiterempfehlen. Außerdem begrüßten die Studierenden die Möglichkeit, in selbst gewählter Geschwindigkeit lernen zu können. Auch deutete sich ein erhöhter Lernerfolg jener Studierenden an, die intensiver mit dem System übten. Bei der Einführung von IDLE-SQL [Pahl und Kenny, 2009] zeigte sich bei gleichbleibenden übrigen Faktoren eine signifikante Verbesserung der Ergebnisse in relevanten Klausuraufgaben gegenüber dem Vorjahr. Zudem stimmten mehr als 90 % der Studierenden der Aussage zu, dass das Trainingssystem ein nützliches Lehr- und Lernwerkzeug sei. Als wichtigsten Aspekt des Systems nannten die Studierenden die Möglichkeit des aktiven statt passiven Lernens. Es folgte der Aspekt der ständigen Verfügbarkeit und die

Möglichkeit der selbst wählbaren Lerngeschwindigkeit. In einem Experiment mit Parallelgruppen schnitt die Gruppe, die das Tutorsystem SQL-ACME [Soler et al., 2006] zur Vorbereitung nutzte, in einem Test besser ab als die Gruppe, die sich mit einem menschlichen Tutor vorbereitete. In einer Umfrage zeigte sich außerdem, dass sich die Studierenden motivierter und besser unterstützt fühlten. Außerdem waren sie stärker davon überzeugt, dass sich ihre Fähigkeiten im Umgang mit der SQL gesteigert hätten.

Um Dozenten zum Einsatz solcher Systeme zu ermutigen, sollten sie möglichst leicht zu installieren und zu bedienen sein. In Abschnitt 4.4 wird daher die Implementierung eines SQL Trainingssystems „ÜPS“ vorgestellt, bei dessen Konzeption besonders viel Wert auf die Trennung technisch-administrativer Aufgaben einerseits und rein inhaltlicher Aufgaben andererseits gelegt wurde. Diese Trennung hat letztlich zum Ziel, auch Studierenden die Möglichkeit zu geben, Aufgaben zu erstellen, so dass diese von Kommilitonen bearbeitet werden können. Diese Möglichkeit besteht derzeit allerdings noch nicht.

Vor der konkreten Beschreibung von „ÜPS“ wird erst das verwendete allgemeine Konzept der Feedback-Generierung für SQL-Anfragen in mehreren Ebenen beschrieben.

4.3. Generierung von Feedback zu SQL-Anfragen in mehreren Ebenen

Da es im Allgemeinen viele syntaktische Varianten einer Anfrage geben kann, die jeweils das korrekte Ergebnis liefern, ist es nicht ausreichend, einen statischen Vergleich mit einer oder mehreren Musterlösungen durchzuführen. Statt dessen werden die SQL Anfragen tatsächlich von einem Datenbank Management System ausgeführt, in welchem die zur Aufgabenstellung gehörende Datenbank hinterlegt ist. Beim Versuch einer Ausführung der Anfrage erhalten die Studierenden Feedback in bis zu drei Ebenen, welches in zwei Schleifen generiert und ausgegeben wird. Das Konzept dieser Schleifen ist dabei analog zu den Feedback-Schleifen im Trainingssystem WARP (siehe Anhang ab Seite 3.2) zu verstehen:

- erste Schleife:
  - erste Ebene: Feedback zur Syntax der Anfrage
- zweite Schleife:
  - zweite Ebene: Feedback zur Semantik der Anfrage durch statischen Vergleich mit Musterlösungen
  - dritte Ebene: Feedback zur Semantik der Anfrage durch Vergleich der Ergebnisse

Die Generierung des Feedbacks der zweiten und dritten Ebene findet also simultan statt. Außerdem erfordert sie eine syntaktisch korrekte Anfrage, so dass hier nur dann Feedback generiert wird, wenn in der ersten Schleife bzw. Ebene keine Fehler vorkommen.

Ein Vorteil dieses Verfahrens ist, dass zur Erstellung einer neuen Aufgabe (bei einer vorhandener Datenbank) lediglich eine Musterlösung und ein entsprechender Text zur Aufgabenstellung, aber kein korrektes Ergebnis angegeben werden muss, da dieses stets dynamisch berechnet wird. Der Aufgabentext ist dabei meist nur eine Verbalisierung der Anfrage der Musterlösung in natürlicher Sprache.

4.3.1. Erste Feedback-Ebene: Syntax der Anfrage

Die Syntax der eingegebenen Anfrage wird überprüft, indem sie im DBMS ausgeführt wird. Bei syntaktischer Unkorrektheit wird die entsprechende Fehlermeldung des DBMS zurückgegeben. Diese kann außerdem analysiert und mit zusätzlichen Informationen aufbereitet werden. Es wird also Feedback der Formen KR, EF und AUC gegeben, wobei sich die Korrektheit bei KR nur auf die Syntax bezieht (siehe Abschnitt 1.4 auf Seite 4).

4.3.2. Zweite Feedback-Ebene: Semantik der Anfrage durch Vergleich mit Musterlösungen

Ist die Anfrage also syntaktisch korrekt, wird eine statische Analyse der Anfrage des Lernenden (Lernerlösung) und einer Musterlösungs-Anfrage durchgeführt. Existieren mehrere syntaktische Varianten von Musterlösungen, wird zunächst mittels der Levenshtein-Distanz [Levenshtein, 1966] ein Ähnlichkeitsmaß zwischen der eingegebenen Anfrage und allen Musterlösungs-Anfragen berechnet. Zur statischen Analyse wird dann die zur Lernerlösung ähnlichste Musterlösung als Referenz verwendet. Dabei wird aus beiden Anfragen jeweils ein sogenannter „Parsebaum“ erstellt. Ein Parser zerlegt dabei die Anfragen rekursiv in ihre Komponenten, so dass eine Baumstruktur entsteht. Deren Knoten bzw. Teilbäume können dann isoliert verglichen werden. Durch diese Analyse werden unter anderem die folgenden Fehler erkannt und ausgegeben:

- das Schlüsselwort (`SELECT`, `UPDATE`, `INSERT` oder `DELETE`) fehlt
- bei einer `SELECT` Anfrage fehlen bestimmte Spalten oder es existieren überflüssige Spalten
- in der `WHERE` Klausel fehlen bestimmte Spalten oder es existieren überflüssige Spalten
- die `ORDER BY` (bzw. `GROUP BY`) Klausel fehlt
- in der `ORDER BY` (bzw. `GROUP BY`) Klausel fehlen bestimmte Spalten oder es existieren überflüssige Spalten

Auch hier umfasst das generierte Feedback die Formen KR, EF und AUC.

4.3.3. Dritte Feedback-Ebene: Semantik der Anfrage durch Ergebnisvergleich

Ist die Anfrage syntaktisch korrekt, wird sie im Kontext der hinterlegten Datenbank ausgeführt. Die Musterlösung wird ebenfalls ausgeführt. Hier spielt es bei mehreren

syntaktischen Varianten der Musterlösung keine Rolle, welche ausgeführt wird, da diese semantisch äquivalent sein müssen. Ergebnisse dieser Anfragen sind stets Tabellen. Bei `SELECT`-Anfragen handelt es sich um die Ergebnistabelle der Anfrage. Bei `INSERT`-, `UPDATE`- und `DELETE`-Anfragen dagegen handelt es sich um die Tabelle, die mit der Anfrage manipuliert werden sollte. Die Tabellen werden nun strukturell (Anzahl an Zeilen und Spalten, Spaltennamen) und inhaltlich (Reihenfolge der Zeilen, konkrete Werte in den Tabellenzellen) verglichen. Sind die Ergebnisse strukturell und inhaltlich identisch, gilt die Aufgabe als korrekt gelöst, ansonsten als nicht korrekt gelöst. Zusätzlich zu einer entsprechenden Rückmeldung erhalten die Lernenden beide Ergebnistabellen als Feedback. Bei nicht korrekt gelösten Aufgaben können sie die Tabellen also vergleichen und entsprechende Fehler erkennen. Außerdem wird diejenige Musterlösungen angezeigt, die in der zweiten Feedback-Ebene als Referenz gewählt wurde. Das generierte Feedback hat hier also die Formen KR, EF, AUC und KCR.

4.4. Das webbasierte Trainingssystem ÜPS

Das beschriebene Verfahren wurde als webbasiertes Trainingssystem ÜPS (Übungsprogramm für SQL) implementiert [Ifland et al., 2014b]. ÜPS stellt Aufgaben im Kontext eines vorgegebenen „Szenarios“. Aus technischer Sicht stellt ein Szenario zunächst eine konkrete SQL Datenbank dar, die mit Daten gefüllt ist. ÜPS verwendet für diese Datenbanken MySQL. Innerhalb eines Szenarios gibt es Aufgabengruppen, die entweder „unbewertete“ oder „bewertete“ Aufgabengruppen sind. Unbewertete Aufgabengruppen dienen ausschließlich zum freien Üben, bei denen Lernende vollständiges Feedback aus allen Ebenen erhalten. Bewertete Aufgabengruppen mit eingeschränktem Bearbeitungszeitraum dienen zum Prüfen der Fähigkeiten im Umgang mit der SQL. Damit können beispielsweise Zwischentests (für den Übungsbetrieb) durchgeführt werden, wobei dazu zwei Feedback-Modi existieren:

- **eingeschränktes Feedback**: Es wird nur das Feedback der ersten Ebene und das Ergebnis der eigenen Anfrage aus der dritten Ebene angezeigt. Bei diesen Aufgaben erhalten Studierende also lediglich Rückmeldung über die syntaktische Korrektheit der Anfrage und darüber, welches Ergebnis das DBMS auf ihre Anfrage zurückliefert bzw. welchen Zustand die Datenbank nach Ausführung der Anfrage hat. Eine semantische Analyse und ein Ergebnisvergleich mit der Musterlösung finden also nicht statt. In diesem Modus wird bezüglich des Feedbacks also das typische Verhalten eines DBMS nachgestellt.
- **ohne Feedback**: Die Studierenden erhalten lediglich eine Bestätigung, dass ihre Anfrage im System gespeichert wurde.

Die Aufgaben werden dennoch automatisch bewertet, wobei eine Rückmeldung über die Bewertung erst nach Ablauf des Bearbeitungszeitraums erfolgt. Zur einfachen Eingabe von Szenarien und zugehörigen Aufgaben wurde ein Autorensystem in die Webanwendung integriert, welches das schnelle und unkomplizierte Erstellen und Modifizieren von

Szenarien und Aufgaben ermöglichen soll. Die Dialoge zur Bearbeitung der Aufgaben werden Lernenden ebenfalls über die Webanwendung präsentiert. Zur Verwaltung und Bewertung der Bearbeitungen der Lernenden existiert ein entsprechendes Administrationssystem.

4.4.1. Technische Aspekte

4.4.1.1. Verwendete Technologien

Die verwendeten Technologien sind clientseitig HTML, JavaScript und CSS. Serverseitig wird Java7 auf Tomcat7 mit den Frameworks *JavaServer Faces*[1] und *Hibernate*[2] verwendet. MySQL Server kommt sowohl als DBMS für die Szenarien-Datenbanken, als auch für die administrativen Datenbanken zum Einsatz.

4.4.1.2. Anbindung an Moodle und Authentifizierung

ÜPS wird in identischer Weise zum Trainingssystem WARP an Moodle oder andere Lernplattformen angebunden (siehe Abschnitt 3.5.1.2 auf Seite 76) und benötigt somit keine eigene Benutzerverwaltung. Eine Rückmeldung der Ergebnisse an die Lernplattform erfolgt über Export bzw. Import von CSV-Dateien aus der bzw. in die entsprechende Datenbank von ÜPS bzw. der Lernplattform.

4.4.1.3. Sicherheit

In ÜPS können beliebige MySQL-Anfragen eingegeben und an das serverseitige DBMS geschickt und dort ausgeführt werden. Dies birgt die Gefahr, dass dabei absichtlich oder unabsichtlich unerwünschte Operationen auf der Datenbank ausgeführt werden oder unerwünschte lesende Zugriffe stattfinden. Dies wird in ÜPS wie folgt verhindert: Für die Webanwendung steht auf dem Server eine eigene DBMS Instanz zur Verfügung. Dies verhindert, dass weitere Anwendungen, die auf dem gleichen Server laufen und MySQL benutzen, durch Fehler in ÜPS beeinflusst werden. Innerhalb einer MySQL Instanz können mehrere Datenbanken erstellt werden, die wiederum mehrere Tabellen enthalten können. Die administrativen Tabellen (Bearbeitungsstatistiken, Metadaten zu Szenarios, Aufgaben und Aufgabengruppen etc.) stehen dabei in einer eigenen Datenbank. Zusätzlich besteht für jedes Szenario eine separate Datenbank. Hier werden für jeden Benutzer bei der Bearbeitung von Aufgaben innerhalb des Szenarios eigene Tabellen angelegt, die strukturell und inhaltlich identisch zu der im Szenario angegebenen Datenbank sind. Ausnahme ist, dass jede Tabelle ein bestimmtes Präfix hat, das einem Benutzer eindeutig zugeordnet werden kann. Anfragen, die vom Studierenden eingegeben werden, werden ausschließlich auf diesen Tabellen ausgeführt. Die Tabellen werden dabei nach

[1] https://javaserverfaces.java.net/ (August 2014)
[2] http://hibernate.org/ (August 2014)

jeder Aufgabe, deren Anfrage eine Tabelle manipuliert, in ihren ursprünglichen Zustand zurückversetzt. Für die Ausführung der Anfragen auf diese Benutzer-Tabellen wird zudem ein eigener MySQL-Benutzer verwendet, der weder Zugriffsrechte auf Datenbanken anderer Szenarien, noch auf die administrative Datenbank hat.

4.4.2. Autorensystem

Um viele Übungsmöglichkeiten für die Studierenden schaffen zu können, wurde ein Autorensystem implementiert, mit dem Szenarien und Aufgaben einfach und unkompliziert angelegt werden können. Das Konzept sieht vor, dass sich Autoren auf die rein inhaltliche Arbeit konzentrieren können und die Administration des DBMS wegfällt. Dessen Installation muss nur einmalig von einem Techniker vorgenommen werden.

4.4.3. Erstellung von Szenarien

Zur Erstellung eines Szenarios muss im entsprechenden Dialog eine Textdatei mit `CREATE` und `INSERT` Statements hochgeladen werden, in welcher Struktur und Daten der Szenario-Datenbank definiert sind. Optional zur Erstellung dieser Textdateien ist die Verwendung von externen Tools wie MySQL Workbench, HeidiSQL oder SQLyog.
Außerdem erhält jedes Szenario einen narrativen Anker im Sinne der *Anchored Instruction* [Bransford et al., 1990]. Dieser soll dazu dienen, Interesse zu wecken und Aufmerksamkeit auf das Wahrnehmen und Verstehen der gestellten Probleme zu lenken [Niegemann et al., 2008]. Optional kann der Autor ein Entity-Relationship-Diagramm bereitstellen, das die Struktur der Datenbank abbildet und den Studierenden vor oder bei Bearbeitung der Aufgaben als Hilfestellung angezeigt wird. Weitere anzugebende Metadaten sind der Name des Szenarios und dessen Start- und End-Daten. Letztere dienen dazu, den Bearbeitungszeitraum eines Szenarios für die Lernenden zeitlich einzuschränken.

4.4.4. Erstellung von Aufgabengruppen

Zur Erstellung von Aufgabengruppen (oder „Übungsblättern“) gibt es einen weiteren Dialog, in dem der Autor deren Namen, deren Typ (also bewertet oder unbewertet), sowie den zulässigen Bearbeitungszeitraum angeben muss (siehe Abbildung 4.1).

4.4.5. Erstellung von Aufgaben

Innerhalb von Aufgabengruppen können nun beliebig viele Aufgaben angelegt werden. Im entsprechenden Dialog müssen der Fragetext, der Schwierigkeitsgrad und eine oder mehrere Musterlösungen eingegeben werden. Der Schwierigkeitsgrad wird Lernenden bei der Bearbeitung der Aufgaben angezeigt und gibt so eine entsprechende Rückmeldung. Bei bewerteten Aufgabengruppen ist die Schwierigkeit außerdem gleichbedeutend mit

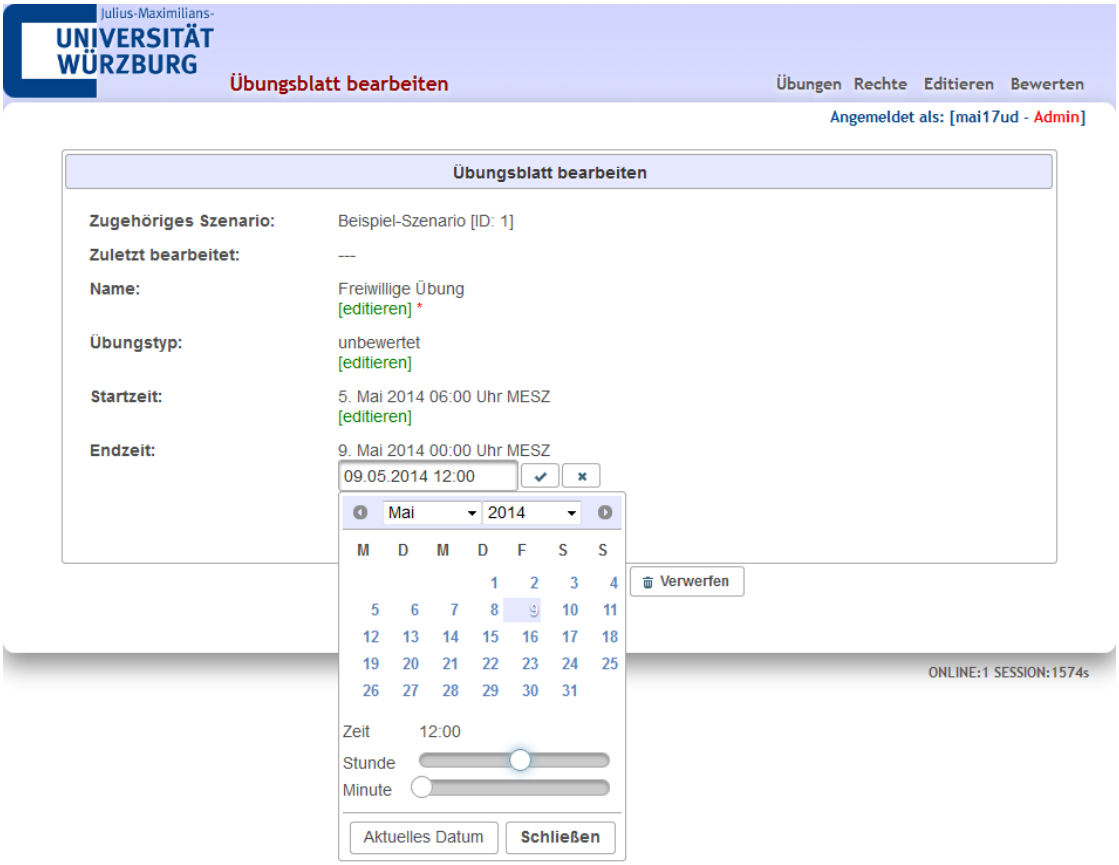


Abbildung 4.1.: ÜPS Autorensystem: Auswahl des Enddatums des zulässigen Bearbeitungszeitraums bei der Erstellung einer Aufgabengruppe.

der Punktzahl, die Lernende für das korrekte Lösen der Aufgabe erhalten. Bei der Eingabe der Musterlösungs-Anfragen können dies unmittelbar auf syntaktische Korrektheit überprüft werden, wobei Sie im DBMS ausgeführt werden. Ist die Anfrage syntaktisch nicht korrekt, wird die entsprechende Fehlermeldung des DBMS ausgegeben. Abbildung 4.2 zeigt die Eingabe einer Aufgabe und eine entsprechende Fehlermeldung.

4.4.6. Bearbeitung von Aufgaben

Ein Link auf die Webanwendung führt die Lernenden stets auf die Portalseite eines bestimmten Szenarios. Dort wird die Beschreibung des Szenarios als narrativer Anker angezeigt, außerdem die zugehörigen Aufgabengruppen bzw. deren Namen (siehe Abbildung 4.3).

Über die Aufgabengruppen können Lernende nun zum Dialog zur Bearbeitung einer Aufgabe navigieren. Hier wird ihnen der Aufgabentext und der entsprechende Schwierigkeitsgrad angezeigt. Als zusätzliche Hilfe können die Inhalte der Tabellen der zugrunde

Abbildung 4.2.: ÜPS Autorensystem: Erstellung einer Aufgabe; eine der Musterlösungs-Anfragen ist syntaktisch unkorrekt, was dem Autor durch eine entsprechende Meldung angezeigt wird

liegenden Datenbank sowie das zum Szenario gehörende Entity-Relationship-Diagramm angezeigt werden. Nach der Eingabe der Anfrage in ein Textfeld kann diese nun überprüft bzw. bei Aufgaben in bewerteten Aufgabengruppen ohne Feedback eingetragen werden. Bei Aufgaben in unbewerteten Aufgabengruppen oder bewerteten Aufgabengruppen mit eingeschränktem Feedback wird dieses im selben Dialog angezeigt, ohne dass die Seite vollständig neu geladen werden muss. Abbildung 4.4 zeigt den Dialog nach der Überprüfung einer Anfrage zu einer unbewerteten Aufgabe.

Bei bewerteten Aufgabengruppen, für die ebenfalls eine automatische Auswertung stattfindet, können die Lernenden nach Beendigung des zulässigen Bearbeitungszeitraums ihre Ergebnisse über die Portalseite des entsprechenden Szenarios aufrufen. Es werden ihnen die kumulierten Punkte für die Übungsgruppe angezeigt, außerdem können alle Aufgaben nun frei, also im Modus einer unbewerteten Aufgabengruppe, bearbeitet werden.

4.4.7. Administration bewerteter Aufgabengruppen (Zwischentests)

Mit bewerteten Übungsgruppen lassen sich Zwischentests durchführen, die auch Prüfungscharakter haben können, wenn der Bearbeitungszeitraum auf ein entsprechendes

Abbildung 4.3.: Die Abbildung zeigt den Startdialog des Szenarios „Amazon Warenhaus“ aus Sicht der Lernenden. Zu verschiedenen SQL-Anfragetypen stehen Aufgaben zur Verfügung.

Intervall beschränkt und der Einsatz der Software in einem Computerraum stattfindet. Möglich sind auch mehrtägige Tests, die an verteilten Orten stattfinden, wobei dabei ein Abschreiben oder Gruppenarbeit natürlich nicht verhindert werden kann.
Die Bearbeitungen der Studierenden können jederzeit in der Administrationsoberfläche angezeigt werden. Hier ist auch ersichtlich, wie die Anfragen jeweils bewertet wurden. Eine Aufgabe gilt als bestanden, sofern die Anfrage syntaktisch korrekt und der Ergebnisvergleich erfolgreich war. Hier werden dem Dozenten vor allem auch solche Bearbeitungen angezeigt, die die statische Analyse nicht bestanden haben, also von den vorgegebenen Musterlösungen abweichen, aber dennoch ein korrektes Ergebnis liefern. Diese Anfragen können dann unkompliziert den Musterlösungen für diese Aufgabe hinzugefügt werden. Weitere Musterlösungen können Aufgaben manuell hinzugefügt werden.
Nach Veränderungen an den Musterlösungen einer Aufgabe (Löschen, Hinzufügen, Ändern), werden sämtliche bisher von den Studierenden eingetragenen Anfragen erneut bewertet. Außerdem können Aufgaben auch manuell nachkorrigiert werden, so dass Bearbeitungen von Aufgaben auch dann als (teilweise) bestanden gewertet werden können, wenn die Anfrage syntaktisch nicht korrekt ist oder ein fehlerhaftes Ergebnis liefert. Auch können Dozenten hier Kommentare zu Bearbeitungen abgeben, die den Studierenden zusammen mit dem Ergebnis der automatischen oder manuellen Korrektur angezeigt

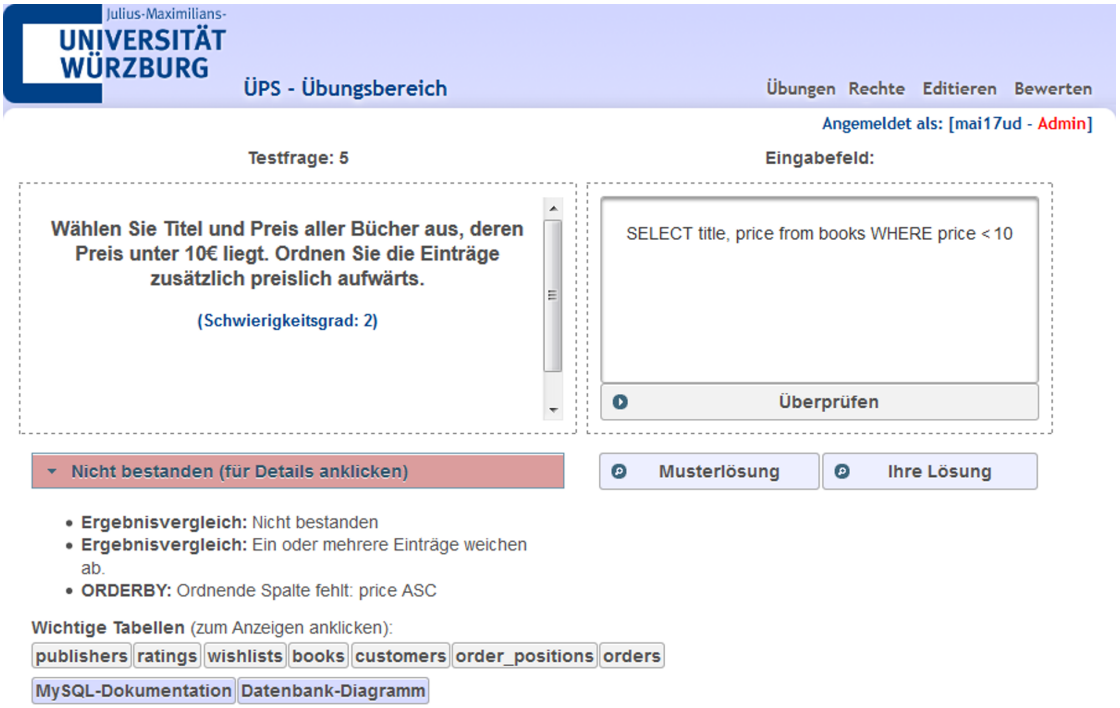


Abbildung 4.4.: Die Abbildung zeigt den Dialogzur Bearbeitung einer Aufgabe. Die eingegebene Anfrage ist hier semantisch nicht korrekt, weshalb entsprechende Fehlermeldungen abgezeigt werden.

werden. Die Ergebnisse einer bewerteten Aufgabengruppe können anschließend als CSV-Datei exportiert werden, um sie beispielsweise in eine Lernplattform zu importieren oder statistische Auswertungen vornehmen zu können.

4.5. Evaluation und Ergebnisse

ÜPS wurde in den Sommersemestern 2013 und 2014 im Übungsbetrieb zur Vorlesung Softwaretechnik an der Universität Würzburg eingesetzt. Die Rahmenbedingungen waren dabei identisch zum Experiment zur Bewertung von UML Klassendiagrammen (siehe Abschnitt 2.6 auf Seite 46). Im Sommersemester 2013 gab es ein Szenario „Amazon Warenhaus“, in welchem insgesamt 61 Aufgaben gestellt wurden, davon 46 in unbewerteten Aufgabengruppen zum freien Üben (Trainingsaufgaben). 15 weitere Aufgaben wurden in bewerteten Aufgabengruppen im Übungsbetrieb angeboten, deren Bearbeitung also verpflichtend war (Testaufgaben). Im Sommersemester 2014 wurden fast alle Aufgaben des „Amazon Warenhauses“ als Trainingsaufgaben angeboten. Zusätzlich gab es ein weiteres Szenario zur „Fußball WM 2010“ (ER-Diagramme siehe Anhang ab Seite 197). Dazu wurden 42 Trainingsaufgaben und 24 Testaufgaben gestellt. Die Bearbeitungszeit

für die Testaufgaben betrug jeweils etwa eine Woche. ÜPS stand den Studierenden auch nach Ablauf der Bearbeitungszeit noch weiterhin zum freien Üben zur Verfügung.

4.5.1. Bearbeitungsstatistiken

4.5.1.1. Sommersemester 2013

Die im Sommersemester eingesetzte Implementierung des Trainingssystems unterstützte einige vorgestellte Funktionen noch nicht. Diese beziehen sich allerdings hauptsächlich auf das Autorensystem und die interne Datenhaltung und sind für die im Folgenden vorgestellten Ergebnisse nicht relevant. Lediglich die dritte Feedback-Ebene für `UPDATE`-, `INSERT`- und `DELETE`-Aufgaben stand im Evaluationsszenario nicht zur Verfügung, da sie nur für `SELECT`-Aufgaben angeboten wurde (vgl.Tabelle 4.3 letzte Zeile). Tabelle 4.2 zeigt Bearbeitungsstatistiken der Aufgaben in unbewerteten und bewerteten Aufgabengruppen. Eine Bearbeitung bezieht sich dabei auf einen Studierenden und eine Aufgabe. Eine Bearbeitung umfasst also mehrere SQL-Anfragen und gilt dann als bestanden, wenn es mindestens eine dieser SQL-Anfragen zu einer der Musterlösungen semantisch äquivalent ist, die Ausführung also das gleiche Ergebnis zurückliefert, was auch syntaktische Korrektheit impliziert.

Aufgaben-Typ	**SELECT**	**UPDATE**	**INSERT**	**DELETE**	**Gesamt**
Anzahl Aufgaben	39/9	2/2	2/2	3/2	46/15
Anzahl Studierende	269/283	158/267	152/268	191/271	270/283
Anzahl Bearbeitungen	5998/2395	277/531	260/532	366/541	6910/3999
Anzahl SQL-Anfragen	18973/8715	547/1057	562/1311	795/1659	20877/12742
Anteil bestandener Bearbeitungen	74 %/72 %	65 %/96 %	46 %/86 %	57 %/88 %	72 %/79 %
Ø Anzahl SQL-Anfragen pro Bearbeitung	3,2/3,6	2,0/2,0	2,2/2,5	2,2/3,1	3,0/3,2
Anteil Bearbeitungen mit korrekter Lösung bei der ersten Anfrage	22 %/46 %	33 %/68 %	7 %/54 %	23 %/57 %	22 %/52 %
Ø Anzahl Anfragen bis zur korrekten Lösung (nur bestandene Bearbeitungen)	2,9/2,4	1,9/1,6	2,8/2,0	2,1/2,2	2,9/2,2

Tabelle 4.2.: Bearbeitungsstatistiken Trainingsaufgaben (linke Werte) und Testaufgaben (rechte Werte) im Sommersemester 2013.

Obwohl die Bearbeitung der Trainingsaufgaben nicht verpflichtend war, wurden fast 21.000 SQL-Anfragen eingegeben. Insgesamt übersteigt die Anzahl der SQL-Anfragen bei freiwilligen Trainingsaufgaben die der verpflichtenden Testaufgaben (etwa 13.000) deutlich. Die Anfragen bei Trainingsaufgaben wurden dabei von 270 verschiedenen Studierenden eingegeben. Dies entspricht etwa 95 % der zu diesem Zeitpunkt am verpflichtenden Übungsbetrieb teilnehmenden Studierenden.

Die späteren Zwischentestergebnisse der Studierenden, die freiwillige Übungsaufgaben bearbeitet haben, wurden mit denen der Studierenden vergleichen, die diese nicht bearbeitet haben. Am Zwischentest haben 283 Studierende teilgenommen. Während die 262 Studierenden, die freiwillige Übungsaufgaben bearbeitet haben, einen Anteil bestandener Bearbeitungen von 79,5 % hatten, kamen die 21 Studierenden, die dieses Angebot nicht wahrgenommen hatten, nur auf 74,8 % bestandene Bearbeitungen. Allerdings sind die Gründe, warum die 21 Studierenden das Angebot nicht wahrgenommen haben, ob sie beispielsweise besonders selbstbewusst oder faul waren, unbekannt.

Das von ÜPS zu jeder SQL-Anfrage generierte Feedback wurde ebenfalls erfasst. Tabelle 4.3 zeigt das Vorkommen von Fehlern der drei Feedback-Ebenen in den zugehörigen Bearbeitungen. Trat ein Fehler einer Ebene bei einer Bearbeitung auf, bedeutet dies also, dass mindestens ein Fehler dieser Ebene bei mindestens einer zugehörigen SQL-Anfrage auftrat.

Aufgaben-Typ	**SELECT**	**UPDATE**	**INSERT**	**DELETE**	**Gesamt**
Anteil Bearbeitungen mit Syntaxfehlern	63 %/39 %	62 %/26 %	91 %/26 %	69 %/36 %	65 %/37 %
Anteil Bearbeitungen mit semantischen Fehlern (aus statischer Analyse)	38 %/40 %	9 %/11 %	8 %16 %	16 %/26 %	34 %/31 %
Anteil Bearbeitungen mit semantischen Fehlern (aus Ergebnisvergleich)	1,9 %/0,4 %	-/-	-/-	-/-	1,9 %/0,4 %

Tabelle 4.3.: Vorkommen von Fehlern in den drei Feedback-Ebenen bei Trainingsaufgaben (linke Werte) und Testaufgaben (rechte Werte) im Sommersemester 2013.

Es zeigt sich, dass der Anteil der Syntaxfehler beim Zwischentest deutlich gesunken ist. Eine mögliche Ursache sind die durch die Bearbeitung der Trainingsaufgaben verbesserten Fähigkeiten im Umgang mit der SQL. Die Fehler aus der semantischen Analyse veränderten sich bei `SELECT`-Aufgaben kaum, vor allem bei `INSERT`- und `DELETE`-Aufgaben stiegen sie jedoch stark an. Das beim Zwischentest fehlende Feedback in dieser Ebene ist eine mögliche Erklärung.

4.5.1.2. Sommersemester 2014

Die im Sommersemester 2014 eingesetzte Implementierung speicherte aus Performanzgründen nicht jede SQL-Anfrage ab, sondern nur die jeweilige aktuellste pro Bearbeitung, so dass keine entsprechenden Zahlen vorliegen.[3] Tabelle 4.4 zeigt die ansonsten analogen Statistiken zu Tabelle 4.2.

[3] Unter anderem diese Maßnahme wurde getroffen, da es im Vorjahr zu Performanzproblemen mit der Datenbank kam.

Aufgaben-Typ	SELECT	UPDATE	INSERT	DELETE	Gesamt
Anzahl Aufgaben	75/18	5/2	3/2	3/2	86/24
Anzahl Studierende	254/263	155/258	127/251	156/257	256/263
Anzahl Bearbeitungen	6217/4477	456/513	155/499	485/512	7313/6001
Anteil bestandener Bearbeitungen	86 %/85 %	84 %/94 %	65 %/96 %	84 %/96 %	85 %/87 %

Tabelle 4.4.: Bearbeitungsstatistiken Trainingsaufgaben (linke Werte) und Testaufgaben (rechte Werte) im Sommersemester 2014.

Hier ist auffällig, dass die `INSERT`-Aufgaben mit etwa 65 % eine wesentlich geringere Bestehensquote aufweisen als die anderen Aufgaben mit etwa 87 %, obwohl die entsprechenden Aufgaben mit einer Schwierigkeit von etwa 1,2 von den Autoren einfacher eingeschätzt wurden als die restlichen Aufgaben mit einer Schwierigkeit von 1,8. Hier wäre es seitens der Autoren bzw Dozenten also ggf. nötig, die Schwierigkeitsgrade anzupassen bzw. in der Präsenzlehre ausführlicher auf diesen Aufgabentyp einzugehen.

Tabelle 4.5, in welcher das Vorkommen von Fehlern nach Feedback-Ebene angezeigt wird, ist anders zu interpretieren als die vermeintlich analoge Tabelle 4.3 des Vorjahres, da sich die Statistiken hier auf nur die jeweils letzte SQL-Anfrage zu jeder Bearbeitung beziehen.

Aufgaben-Typ	SELECT	UPDATE	INSERT	DELETE	Gesamt
Anteil Bearbeitungen mit Syntaxfehlern	11 %/4 %	11 %/2 %	34 %/1 %	12 %/0 %	12 %/3 %
Anteil Bearbeitungen mit semantischen Fehlern (aus statischer Analyse)	4 %/6 %	1 %/1 %	1 %/0 %	4 %/3 %	4 %/5 %
Anteil Bearbeitungen mit semantischen Fehlern (aus Ergebnisvergleich)	1 %/5 %	3 %/4 %	0 %/3 %	0 %/0 %	1 %/5 %

Tabelle 4.5.: Vorkommen von Fehlern in den drei Feedback-Ebenen bei Trainingsaufgaben (linke Werte) und Testaufgaben (rechte Werte) im Sommersemester 2013. Pro Bearbeitung wurde hier nur die aktuellste SQL-Anfrage gespeichert.

Es handelt sich hier also um die Fehlerverteilung der finalen Einreichung zu einer Aufgabe, die vor allem bei den verpflichtenden Testaufgaben interessant ist. So zeigt sich, dass bei etwa 3 % der Aufgabenbearbeitungen syntaktisch fehlerhafte SQL-Anfragen nicht verbessert, sondern fehlerhaft eingereicht wurden, obwohl den Studierenden eine entsprechende Fehlermeldung vorlag. Dafür kann es mehrere Gründe geben. Eine Möglichkeit ist, dass Studierende ihre anderen Einreichungen als gut genug einschätzen, um ausreichend viele Punkte zu erhalten. Eine andere Möglichkeit ist, dass sie es schlicht nicht geschafft haben, eine syntaktisch korrekte Anfrage einzugeben. Bei insgesamt etwa 10 %

der Einreichungen zu den Testaufgaben lagen semantische Fehler vor, die sich gleichmäßig auf Fehler durch den statischen Vergleich und auf Fehler durch den Ergebnisvergleich aufteilen.

4.5.2. Umfrageergebnisse

Nach der Beendigung der Experimente wurden jeweils Befragungen der Studierenden durchgeführt.

4.5.2.1. Sommersemester 2013

Etwa eine Woche nach Bearbeitung der Aufgabe wurde den Studierenden ein schriftlicher Fragebogen vorgelegt (siehe Anhang Seite 181), in welchem auch konkrete Fragen zur Verwendung von ÜPS gestellt wurden. Die Studierenden sollten bestimmte Aspekte der Anwendung mit Schulnoten (1-6) bewerten. Tabelle 4.6 zeigt die Ergebnisse dieser Umfrage.

Aspekt	**Ø Note**
Konzept des Tools, unabhängig von der Implementierung	1,6
Umsetzung des Tools, Gesamtnote	2,05
Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)	1,77
Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)	3,43
Aspekt Inhalt der Übungsaufgaben	1,83
Aspekt Fairness der Bewertung der Übungsaufgaben	1,94
Aspekt hilfreiche Erklärungen als Text oder nützliches Feedback	2,28

Tabelle 4.6.: Umfrageergebnisse zur Verwendung von ÜPS im Sommersemester 2013; die verschiedenen Aspekte wurden mit Schulnoten (1-6) bewertet (n=240)

4.5.2.2. Sommersemester 2014

Bei Bekanntgabe der Verfügbarkeit der Ergebnisse der verpflichtenden Testaufgaben per E-Mail wurden die Studierenden gebeten, im schon zuvor vorgestellten Online-Fragebogen die hinzugefügten Fragen zur Verwendung von ÜPS zu beantworten (vollständiger Fragebogen siehe Anhang Seite 182). Tabelle 4.7 zeigt die Ergebnisse dieser Umfrage. Die Frage, ob ÜPS in Zukunft auch weiterhin im Übungsbetrieb eingesetzt werden soll, beantworteten 88 % der Studierenden mit „Ja“.

Aufgrund des geringen Rücklaufs bei den in der Umfrage enthaltenen Freitextfragen ($n = 5$) wurde auf deren systematische Evaluierung verzichtet. Als positive Aspekte wurden vor allem die freie Möglichkeit zum Üben genannt, was sich auch mit den entsprechenden Bearbeitungsstatistiken deckt. Als negative Aspekte wurden Performanzprobleme genannt.

Aspekt	Ø Note
Konzept des Tools, unabhängig von der Implementierung	1,41
Umsetzung des Tools, Gesamtnote	1,82
Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)	1,82
Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)	1,88
Aspekt Inhalt der Übungsaufgaben	1,94

Tabelle 4.7.: Umfrageergebnisse zur Verwendung von ÜPS im Sommersemester 2014; die verschiedenen Aspekte wurden mit Schulnoten (1-6) bewertet (n=17)

4.5.2.3. Diskussion

In den Umfragen wurde ÜPS in fast allen Aspekten durchschnittlich mit der Note „gut“ bewertet. Ausnahme ist der Aspekt „Technik“ im Sommersemester 2013. Eine mögliche Erklärung dazu ist, dass es während des Bearbeitungszeitraums zu einem hardwarebedingten Ausfall des kommerziellen Servers kam und das System so für einige Stunden nicht erreichbar war. Des weiteren traten gerade bei vielen simultanen Zugriffen sehr hohe Reaktionszeiten der Datenbank auf. Für diese Gründe spricht auch, dass die Technik im folgenden Semester gut bewertet wurde, wo es nicht zu einem Serverausfall kam und ein effizienteres Datenbankdesign gewählt wurde.
Der hohe Prozentsatz für den zukünftigen Einsatz des Tools und die Tatsache, dass die beste Note mit 1,6 für das Konzept von ÜPS, unabhängig von der Implementierung, gegeben wurde, zeigen, dass noch Verbesserungspotential bei der konkreten Implementierung besteht.

4.6. Zusammenfassung und Ausblick

Die Ergebnisse zeigen, dass ÜPS effektiv einsetzbar ist und von den Studierenden gut angenommen wurde. Die Bearbeitungsstatistiken zeigen, dass die Studierenden vor allem mit der Syntax der SQL Probleme haben. Bei fast zwei Drittel aller Bearbeitungen der Trainingsaufgaben traten Syntaxfehler auf. Dies zeigt, dass ein mehrstufiger Feedback-Ansatz sinnvoll ist, da ohne syntaktische Korrektheit die Semantik nicht überprüft werden kann. Bei den Trainingsaufgaben benötigten die Studierenden dabei durchschnittlich etwa drei Iterationen, um die Aufgabe zu lösen. Auffällig ist, dass nur in knapp 2 % der Bearbeitungen von Trainingsaufgaben (des Typs `SELECT`) Fehler aus der dritten Feedback-Ebene aufgetreten sind. Dies legt nahe, dass durch die semantische Analyse in der zweiten Ebene semantische Fehler bereits sehr gut erkannt werden.

Die Evaluationen von ÜPS zeigen, dass die Studierenden die Trennung der Bearbeitung von beliebig vielen freiwilligen Übungsaufgaben mit sofortigem Feedback und einigen Übungsblättern mit Abgabefristen und Feedback am Ende der Abgabefrist, mit denen der Leistungsstand erhoben wird, in der großen Mehrheit schätzen. ÜPS zeichnet sich durch die Kombination beider Aspekte aus.

Die in Abschnitt 4.4.2 vorgestellte Trennung technisch-administrativer einerseits und rein inhaltlicher Aufgaben andererseits ermöglicht den Einsatz von ÜPS für didaktisch interessante Konzepte. Nach der Bereitstellung der Webanwendung inklusive der DBMS-Instanz für eine Schule bzw. Hochschule durch einen Techniker, können nicht nur Lehrer bzw. Dozenten, sondern auch Schüler bzw. Studierende, die keine Kenntnisse über die Administration von DBMS haben, konkrete Szenario-Datenbanken (ähnlich wie in eledSQL [Ufert et al., 2013]) selbst erstellen. Zusätzlich können sie mit ÜPS auch Aufgaben erzeugen, die dann von ihren Mitschülern bzw. Kommilitonen gelöst werden müssen. Bei Verwendung bewerteter Aufgabengruppen können die Autoren der Aufgabe die zugehörigen Bearbeitungen anschließend manuell nachkorrigieren und kommentieren. Bei der Verwendung von ÜPS mit diesem Ansatz würden sich neue Lernziele ergeben, die nicht nur in den unteren drei Ebenen der Taxonomie von Bloom [Bloom et al., 1956] liegen, sondern sich auch in die Bereiche „analysieren“, „evaluatieren“ und „erschaffen“ [Anderson und Krathwohl, 2001] erstrecken. Eine Implementierung dieser Erweiterung und entsprechende Evaluierungen sind also die nächsten Ziele.

5. Markierung von Bildregionen durch Freihandzeichnen

Ein Beispiel für eine strukturierte, halb-offene Aufgabe, ist das Markieren von Regionen in Bildern durch Freihandzeichnen. Eine Region bezeichnet dabei eine zusammenhängende Fläche, also ein (nicht überschlagenes) Polygon.

5.1. Beschreibung des Aufgabentyps und verwandte Arbeiten

Dem Lernenden wird ein Bild angezeigt und er wird angewiesen eine oder mehrere bestimmte Regionen zu markieren. Es ist also keine Menge von bereits markierten Regionen vorhanden, aus denen Lernende eine oder mehrere selektieren müssen, was in Rütters Aufgabentypologie [Rütter, 1982] einer geschlossenen Identifikationsaufgabe entspräche. Stattdessen müssen Lernende die Regionen selbstständig konstruieren. Dies geschieht bei elektronischer Bearbeitung mittels verbreiteter Mausgesten, beispielsweise durch das „Aufziehen“ der Region durch Bewegung der Maus bei gedrückter linker Taste. Aufgaben diesen Typs werden im Folgenden auch als „Bildmarkierungsaufgaben“ bezeichnet.

Bei solchen Bildmarkierungsaufgaben handelt es sich um halb-offene Aufgaben: Dem Auswerter liegt eine korrekte Lösung vor. Den Lernenden liegt diese Lösung zwar auch vor, jedoch existiert eine sehr große Menge möglicher Antworten, die eingeben werden können. Es besteht diesbezüglich also eine Analogie zu Long-Menu-Fragen. Zudem handelt es sich um einen strukturierten Aufgabentyp, da die Antwort letztlich nur aus einer Menge ausgewählter Bildpunkte[1] besteht, die sich formal sehr einfach darstellen lässt.

Einsatzgebiete existieren vor allem in der Medizin, da bildgebende Verfahren (Röntgen, MRT, Ultraschall etc.) in der Diagnostik weit verbreitet sind. Mögliche konkrete Aufgaben sind unter anderem die Markierung von...

- Onkologie
  - ...Tumoren in MRT-Bildern
  - ...Tumoren in Röntgenbildern
  - ...Tumoren in Ultraschallbildern
  - ...Melanomen in Fotografien
- Zahnmedizin

[1] Es wird bei diesem Ansatz von Rastergrafiken ausgegangen.

  - ... kariösen Stellen in Fotografien
  - ... Konkrementen (Zahnstein) in Fotografien
- Neurologie
  - ... durch Morbus Alzheimer veränderte Hirnregionen in MRT-Bildern
- Kardiologie
  - ... Auffälligkeiten im Elektrokardiogramm (EKG)

In der Literatur werden Bildmarkierungsaufgaben auch als *HotSpot-* oder *ImageMap-*Aufgaben bezeichnet. Dabei werden die Lernenden angewiesen, bestimmte Regionen im vorgegebenen Bild zu markieren, wobei es verschiedene Variationen gibt. Aufgaben mit bereits sichtbar vorgegebenen Regionen sind eine letztlich nur einer Spezialform von Multiple-Choice-Aufgaben [Vogt und Schneider, 2009], bei der die Antwortalternativen grafisch dargestellt werden. Auch [Schmees et al., 2013] sprechen von Bildmarkierungsaufgaben, wobei jedoch unklar bleibt, ob auch hier lediglich relativ große Bildsegmente selektiert oder ob Regionen selbst konstruiert werden (siehe Abbildung 5.1). In der von der Justus-Liebig-Universität Gießen betriebenen Lernplattform k-MED können ebenfalls Bildmarkierungsaufgaben erstellt und bearbeitet werden [Wagner et al., 2006]

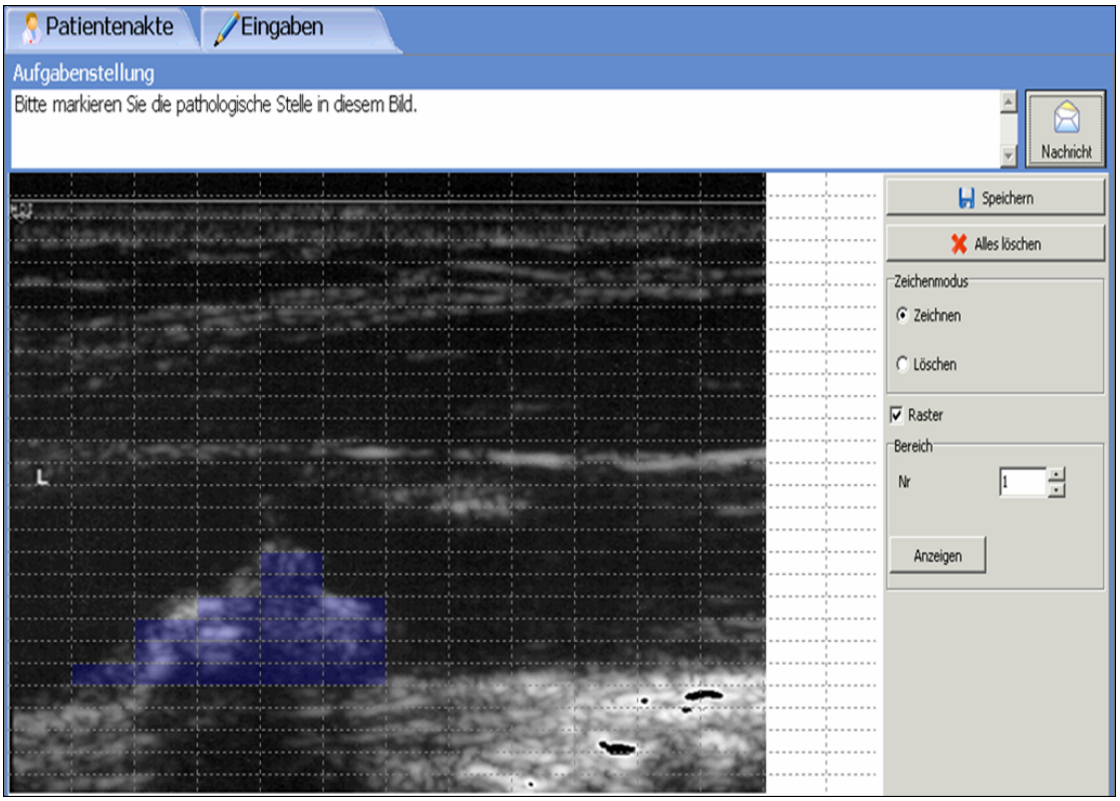


Abbildung 5.1.: Bildmarkierungen auf einem Röntgenbild [Schmees et al., 2013, S. 7]

Es finden sich allerdings keine klaren Hinweise auf den Einsatz von Bildmarkierungsaufgaben, bei denen Lernende die Regionen selbst detailliert konstruieren müssen, also

beispielsweise durch Freihandzeichnen. Diese Eingabemethode eignet sich jedoch gerade für solche Bildmarkierungsaufgaben, bei denen eine genaue Abgrenzung von „korrekten“ zu „inkorrekten“ Regionen wichtig ist.

5.2. Integration in das Trainingssystem CaseTrain

Für eine Fallstudie wurde nun verschiedene Komponenten entwickelt, um Bildmarkierungsaufgaben mit Freihandzeichnen in das fallbasierte Trainingssystem CaseTrain zu integrieren [Ifland et al., 2010]. Die in CaseTrain vorhandene Statistik-Komponente erlaubte außerdem eine umfangreiche Evaluation, die am Beispiel eines Trainingsfalles aus dem Fachbereich der interdisziplinären Onkologie durchgeführt wurde.

5.2.1. Autorenwerkzeug

CaseTrain verfolgt bei der Eingabe von Trainingsfällen einen dokumentbasierten Ansatz. Das bedeutet, dass pro Trainingsfall ein einziges Dokument besteht, in dem alle Fallinhalte vorhanden sind. Bei CaseTrain sind dies speziell formatierte Standard Office Text- oder Tabellen-Dokumente, die zur Erstellung des Falles in einem Web-Interface hochgeladen und serverseitig von einem Fall-Parser in ein internes Format konvertiert werden. Lediglich Multimedia-Elemente (Video, Audio, Bild) werden referenziert und dem Dokument beigelegt, wobei im Falle der Benutzung eines Text-Dokuments als Fall-Dokument auch die Möglichkeit besteht, die Bilder direkt in das Text-Dokument einzubinden.

Bei Bildmarkierungsaufgaben muss natürlich in jedem Falle ein Bild eingegeben werden, nämlich das, in welches die Lernenden bei der Ausführung der Aufgabe die Regionen einzeichnen müssen. Um später eine automatisierte Bewertung durchführen und entsprechendes Feedback geben zu können, müssen dem System auch die korrekten Regionen mitgeteilt werden. Für deren Eingabe gibt es verschiedene Ansätze. Eine Möglichkeit wäre eine direkte Eingabe der Koordinaten der Eckpunkte der entsprechenden Polygone bezüglich einer der Ecken des Bildes. Da die Regionen sich meist aber nicht als einfache Polygone mit wenigen Ecken darstellen lassen, wäre diese Form der Eingabe relativ umständlich. Im Hinblick auf die Gebrauchstauglichkeit des Autorenwerkzeugs ist es erstrebenswert, auch die Eingabe der Bildregionen für den Autor möglichst einfach zu gestalten.

CaseTrain löst dies, indem neben dem Bild, welches den Lernenden in der Aufgabe präsentiert wird, ein zweites, identisches Bild angegeben wird, in welchem zusätzlich die Regionen korrekt eingezeichnet sind. Dazu kann der Autor eine ihm geläufige Grafiksoftware benutzen. Der Fall-Parser kann dann aus diesen beiden Bildern die Regionen berechnen. Damit der Fall-Parser erkennen kann, welche Teile des Bildes markierte Regionen darstellen, muss die linke obere Ecke des Bildes in gleicher Farbe wie die der Regionen übermalt werden. Dieses Polygon wird vom Fall-Parser nicht in eine zu markierende Region umgewandelt, sondern dient lediglich der Definition der Markierungsfarbe.

Werden mehrere Regionen eingegeben, ist es nötig, diese zu identifizieren. Dazu muss der Autor mit Hilfe der Grafiksoftware (z. B. mittels eines Textfeldes) eine Zahl in die Region eingeben. Diese wird vom Fall-Parser mittels *Optical Character Recognition* (OCR) erkannt und dient im Parsing-Prozess als Markierungsnummer für die entsprechende Region.

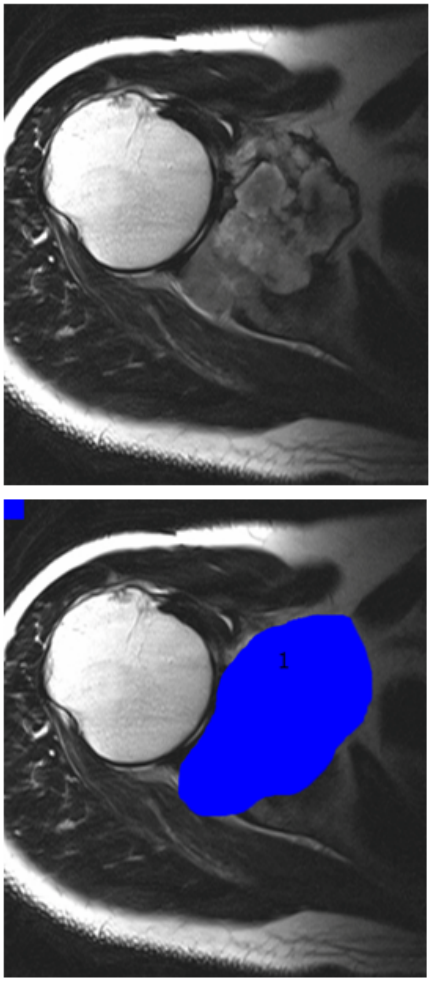
Frage	{1-GM}Markieren Sie die tumoröse Läsion auf dem Bild (mit einer Markierung)!?
Vorlage	
Antwort	{1: 1-0.2-0.7 +2}

Abbildung 5.2.: Eingabe einer Bildmarkierungsaufgabe im Autorensystem von CaseTrain. Neben dem rohen Bild wird ein weiteres angegeben, in welchem die zu markierenden Regionen eingezeichnet sind

Abbildung 5.2 zeigt einen Ausschnitt aus einem Office Textdokument zu Fall-Eingabe. Das Dokument ist, den Anforderungen des CaseTrain Autorensystems entsprechend, tabellarisch mit zwei Spalten aufgebaut. In der linken Seite befindet sich jeweils ein

vom System vorgegebenes Schlüsselwort, in der rechten Spalte der jeweilige Inhalt. Die inhaltsdefinierende Zelle in der Zeile „Frage“ beginnt mit in geschweiften Klammern eingeschlossenen Optionen zu dieser Frage. Die „1“ steht hierbei für die Gewichtung dieser Aufgabe im Fall, „GM“ bezeichnet den Typ der Aufgabe (**G**rafik**m**arkierungsaufgabe). Es folgt der Fragetext, der den Lernenden bei der Bearbeitung des Falles angezeigt wird, anschließend die Zeile mit dem Schlüsselwort „Vorlage“, in der die beiden weiter oben beschriebenen Bilder stehen.

Um die von Lernenden gezeichneten Regionen später automatisch bewerten zu können, müssen für jede Region noch weitere Daten eingegeben werden. Dies geschieht in der Zeile „Antwort“ in folgender Syntax: `{M: W-Su-So +G}`

`M` entspricht dabei der **M**arkierungsnummer, `W` steht für die Ge**w**ichtung dieser Region bei der Bewertung dieser Aufgabe im Vergleich zu anderen Regionen. Für die Markierung jeder Region wird, wie weiter unten genauer beschrieben, ein Prozentwert berechnet, der den *Überdeckungsgrad* mit der Musterlösung angibt. Mit S_u und S_o lassen sich untere und obere Schranken für die letztliche Bewertung des Überdeckungsgrades dieser Region festlegen. Die untere Schranke gibt an, bis zu welchem Überdeckungsgrad die Markierung dieser Region mit **n**ull Prozent, die obere Schranke ab welchem Überdeckungsgrad mit **h**undert Prozent bewertet wird. Eine Eingabe von beispielsweise S_u=0,2 und S_o=0,7 bedeutet, dass die Markierung dieser Region bei einem Überdeckungsgrad von unter 20 % mit 0 % und bei einem Überdeckungsgrad von über 70 % mit 100 % bewertet wird. `G` entspricht dem „Straf-Faktor“ für zu **g**roße Markierungen. Eine genaue Erläuterung dazu folgt in Abschnitt 5.2.3 über die automatische Auswertung und Feedback-Generierung.

5.2.2. Aufgabenbearbeitung

Um die Aufgabe im CaseTrain-Player bearbeiten zu können, wurde dieser um eine entsprechende Komponente erweitert. Die Aufgabe wird dem Lernenden wie eine gewöhnliche Frage präsentiert (siehe Abbildung 5.3).

Analog zum Editor für UML Klassendiagramme (siehe Abschnitt 2.4.3 auf Seite 39) wird den Lernenden das Freihandzeichen-Werkzeug zur Markierung in einem modalen Dialog angezeigt, der durch das Betätigen der Schaltfläche „Markieren“ geöffnet wird. Dort wird den Lernenden das Bild in voller Auflösung dargestellt. Das Markieren der Region geschieht durch das Ziehen des Mauszeigers bei gedrückter linker Maustaste, wobei das Polygon nach einer Mausbewegung unmittelbar neu gezeichnet und eingefärbt wird (siehe Abbildung 5.4). Die Farbe ist dabei identisch zu der vom Autor im Rahmen der Fallerstellung benutzen Farbe. Ein Polygon kann nur in einem Zug gezeichnet werden, nachträglich ist es also nicht mehr veränderbar. Um Fehler zu korrigieren, muss das Zeichnen des gesamten Polygons durch einen Klick auf die entsprechende Schaltfläche (links oben) rückgängig gemacht werden.

Wenn das Markieren von Regionen abgeschlossen ist, kann der Dialog wieder geschlossen werden. Die Markierungen werden den Lernenden dann auch in der verkleinerten Ansicht im Antwort-Bereich angezeigt und können jederzeit überarbeitet werden, solang die aus

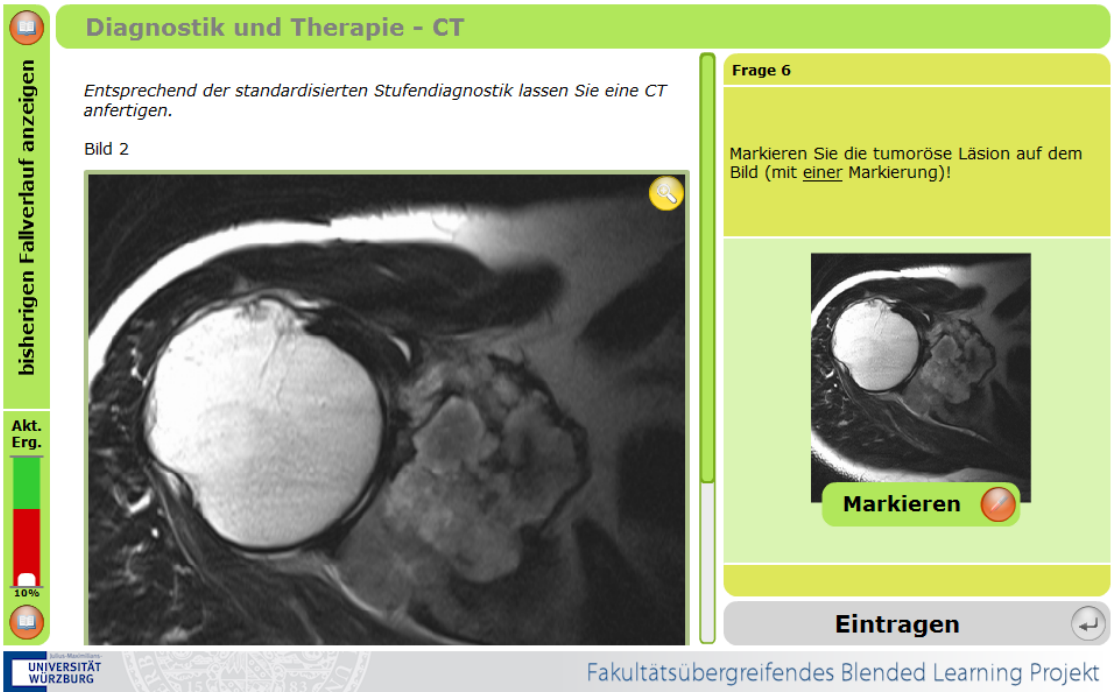


Abbildung 5.3.: Präsentation einer Bildmarkierungsaufgabe im CaseTrain-Player

den Markierungen bestehende „Antwort“ noch nicht eingetragen wurde. Geschieht dies über die entsprechende, zuvor ausgegraute Schaltfläche, wird die Bewertung für diese Aufgabe berechnet und ein entsprechendes Feedback wird generiert.

5.2.3. Bewertung und Feedback-Generierung

Die vom Autor eingegebenen Polygone werden im Folgenden als Polygone der *Musterlösung* bezeichnet, die von Lernenden eingegebenen Polygone als die der *Lernerlösung*. Zudem werden die Begriffe *Region* und *Polygon* synonym verwendet.

5.2.3.1. Bewertung

Die Bewertung der Regionen wird über die Berechnung der Schnittflächen von Polygonen der Muster- und Lernerlösung berechnet. Betrachtet man den Schnitt zweier Polygone `M` und `L` in einer Ebene, ergeben sich vier Teilflächen. Sie enthalten jeweils Punkte, die...

A) ...in beiden Polygonen liegen
B) ...weder in Polygon `M`, noch in Polygon `L` liegen
C) ...nur in Polygon `M` liegen
D) ...nur in Polygon `L` liegen

Abbildung 5.4.: Markieren einer Region bei einer Bildmarkierungsaufgabe im CaseTrain-Player

Aus den Flächeninhalten dieser Teilflächen wird nun der Überdeckungsrad berechnet, wobei letztlich analog zur Auswertung von Multiple-Choice-Fragen vorgegangen wird, bei welchen Lernende durch die Auswahl oder Nicht-Auswahl der vorgegebenen Antworten stets eine binäre Klassifikation vornehmen.

Bei Multiple-Choice-Fragen gibt es Antworten, die...

A) ...korrekt sind und ausgewählt wurden (**R**ichtig **P**ositive)
B) ...nicht korrekt sind und nicht ausgewählt wurden (**R**ichtig **N**egative)
C) ...korrekt sind und nicht ausgewählt wurden (**F**alsch **N**egative)
D) ...nicht korrekt sind und ausgewählt wurden (**F**alsch **P**ositive)

Üblicherweise werden Lernende für RP Antworten belohnt und für FP Antworten bestraft. Es existieren auch Auswertungsverfahren, bei denen zusätzlich für RN Antworten belohnt und FN Antworten bestraft wird. Die Analogie zur Berechnung des Überdeckungsgrades wird deutlich, wenn man sich jeden Pixel des zugrunde liegenden Bildes als Multiple-Choice-Antwort vorstellt, für die Lernende jeweils entscheiden, ob sie sie auswählen oder nicht. Abbildung 5.5 verdeutlicht den Zusammenhang optisch.

Für die Berechnung des Überdeckungsgrades wurde ein möglichst einfaches Auswertungsverfahren gewählt, damit die entsprechende Erklärung für Lernende leichter nachzuvollziehen ist: Die Fläche RP wird positiv, die Fläche FP negativ bewertet. Der Über-

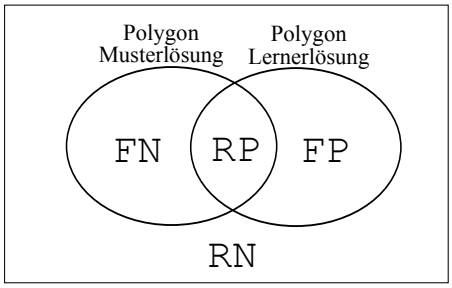


Abbildung 5.5.: Zusammenhang zwischen dem Schnitt zweier Polygone in der Ebene und binärer Klassifikation

deckungsgrad U zwischen zwei Polygonen M (Musterlösung) und L (Lernerlösung) wird dann wie folgt berechnet:

$$U(M,L) = \frac{A(RP(M,L)) - A(FP(M,L)) * SF}{A(M)} \tag{5.1}$$

A steht dabei jeweils für den Flächeninhalt, so dass $A(RP(M,L))$ den Flächeninhalt der RP Fläche und $A(FP(M,L))$ den der FP Fläche der Polygone M und L bezeichnet (siehe Abbildung 5.5). Dabei wird der Abzug für die FP Fläche noch mit einem Faktor SF versehen, über den Autoren regulieren können, wie viel Abzug Lernende für zu großzügiges Markieren von Regionen erhalten. Dies können sie, wie in Abschnitt 5.2.1 beschrieben, bei der Aufgabenerstellung über den „Straf-Faktor“ tun. Durch die Normierung über $A(M)$, also die Fläche des Musterlösungs-Polygons, wird der Überdeckungsgrad normiert, so dass er 100 % nicht überschreiten kann. Nach unten wird der Überdeckungsgrad bei 0 % gedeckelt, so dass er stets zwischen 0 % und 100 % liegt. Für diese Rechnung ist es zudem unerheblich, ob die Fläche anhand von Pixeln, cm^2 oder anderen Einheiten gemessen wird.

Es wird nun für das erste Polygon aus der Musterlösung das Polygon aus der Lernerlösung mit dem höchsten Überdeckungsgrad ausgewählt und so zugeordnet. Dies wiederholt sich für alle weiteren Polygone der Musterlösung, wobei bereits zugeordnete Polygone der Lernerlösung nicht weiter beachtet werden. Ist die Anzahl an Musterlösungs- und Lernerlösunges-Polygonen identisch, ergibt sich so eine eineindeutige Zuordnung zwischen beiden Polygon-Mengen.

Diese Berechnung ist also eine Variation des in Abschnitt 1.6.1 vorgestellten Überdeckungsmaßes. Die Polygone beider Mengen werden einander zugeordnet, wobei dieses Zuordnungsproblem hier nicht durch die Ungarische Methode, sondern durch einen sogenannten *Greedy*-Algorithmus (engl. *greedy* = gierig) gelöst wird. Der Vorteil liegt hier in einer simpleren Implementierung und kürzeren Laufzeit, allerdings im Allgemeinen auf Kosten der Optimalität der Lösung.

Die Bewertung der gesamten Aufgabe basiert auf der Addition der Bewertungen für alle Regionen der Musterlösung, genauer gesagt für die Bewertungen der Überdeckungsgrade

aller Polygone der Musterlösungen und der jeweils zugeordneten Polygone der Lernerlösung. Grundsätzlich ist eine solche einzelne Bewertung identisch zum jeweiligen Überdeckungsgrad. Da es aber für die Lernenden kaum möglich ist, die gewünschten Regionen pixelgenau zu markieren, besteht für die Autoren die in Abschnitt 5.2.1 vorgestellte Möglichkeit, eine obere Schranke zu benutzen. Sie dient eben dazu, eine nicht pixelgenau korrekte Lösung dennoch als vollständig richtig zu bewerten. Die untere Schranke dient dazu, Markierungen mit nur wenig Überschneidung vollständig abzulehnen. Damit kann beispielsweise abgefangen werden, dass Lernende bei der Markierung einer nicht korrekten Region Pixel einer korrekten Region nur zufällig markieren. Liegt der Überdeckungsgrad also unter der unteren bzw. über der oberen Schranke, wird er mit 0 % bzw. 100 % bewertet. Liegt er zwischen den Schranken, wird linear interpoliert, so dass sich für die Bewertung B des Überdeckungsgrades U zweier Polygone in Abhängigkeit von unterer Schranke S_u und oberer Schranke S_o folgende Berechnung ergibt:

$$B(U, S_u, S_o) = \begin{cases} 0\,\% & U <= S_u \\ 100\,\% & U >= S_o \\ \frac{U - S_u}{S_o - S_u} & sonst \end{cases} \tag{5.2}$$

Beispiel: Für einen Überdeckungsgrad von $U = 60\,\%$, eine untere Schranke von $S_u = 20\,\%$ und eine obere Schranke von $S_o = 70\,\%$ ergibt sich eine Bewertung von $B(U, S_u, S_o) = 80\,\%$. Diese Praxis mit oberen und unteren Schranken ist auch bei Multiple-Choice-Fragen üblich.

Für die Bewertung der gesamten Lernerlösung L werden nun, wie bereits erwähnt, die Bewertungen aller Regionen der Musterlösungen aufaddiert. Dabei wird noch die Gewichtung der Region G_N (siehe Abschnitt 5.2) einbezogen:

$$B(L) = \frac{\sum_{n=1}^{N} G_n * B(U(M_n, L_n))}{\sum_{n=1}^{N} G_n} \tag{5.3}$$

N steht dabei für die Anzahl der Polygone der Musterlösung, über die summiert wird. M_n ist das n-te Polygon der Musterlösung, L_n das diesem zugeordnete Polygon der Lernerlösung. G_n entspricht der Gewichtung des n-ten Polygons der Musterlösung. Dies bedeutet, dass die Lernenden für Polygone aus der Musterlösung, denen keine Polygone aus der Lernerlösung zugeordnet werden konnten, keine Prozentpunkte erhalten. Allerdings werden sie auch nicht dafür bestraft, wenn sie mehr Polygone angeben, als in der Musterlösung vorkommen.

5.2.3.2. Feedback-Generierung

Nachdem Lernende ihre Regionen als Antwort eingetragen haben, werden ihnen die korrekten Regionen aus der Musterlösung in verkleinerter Ansicht im Antwortbereich als Kontur angezeigt. Beim Überfahren dieses Bereichs mit dem Mauszeiger sehen sie

zudem die der jeweiligen Region zugeordnete Region aus der Lernerlösung, also die selbst gezeichneten Polygone (siehe Abbildung 5.6). Es wird also Feedback der Form KCR gegeben (siehe Abschnitt 1.4 auf Seite 4).

Abbildung 5.6.: Anzeige der korrekt markierten Region (Musterlösung) im Antwortbereich nach Bearbeitung einer Bildmarkierungsaufgabe ohne (links) und mit (rechts) Einblendung der Markierung der Lernerlösung

Über die Schaltfläche „Erklärung“ können sich Lernende im CaseTrain-Player eine ausführliche Erläuterung über die Berechnung ihrer Bewertung sowie weiterführende Erklärungen in einem modalen Dialog anzeigen lassen. Bei Bildmarkierungsaufgaben werden den Lernenden nochmals alle Regionen der Musterlösungen und optional die eigenen Markierungen simultan angezeigt. Des Weiteren wird die oben beschriebene Berechnung für jede Region der Musterlösung verbal erläutert, womit also Feedback der Formen KP und EF gegeben wird (siehe Abbildung 5.7).

Wie bei offenen und halb-offenen Aufgaben in Trainingsfällen des CaseTrain-Systems üblich, können die Lernenden hier auch eine abweichende Bewertung für ihre Bearbeitung eingeben, falls sie nicht mir der automatischen Bewertung einverstanden sind. Außerdem können sie Kritik in Form eines Freitexts äußern. Diese abweichenden Bewertungen werden zusammen mit der Kritik an Tutoren oder den zuständigen Dozenten übermittelt, so dass eine mögliche fehlerhafte Parametrisierung der Aufgabe korrigiert oder den Lernenden die Bewertung genauer erläutert werden kann. Von diesen Möglichkeiten wurde in dem im Folgenden beschriebenen Experiment in nur sehr geringem Umfang Gebrauch gemacht, sodass von einer systematischen Evaluierung abgesehen wurde.

Abbildung 5.7.: Ausführliches Feedback bei einer Bildmarkierungsaufgabe; Lernende sehen Markierungen aus Muster- und Lernerlösung sowie eine verbale Erläuterung der Bewertung

5.2.4. Evaluation und Ergebnisse

In der medizinischen Fakultät der Universität Würzburg wurde vom Wintersemester 2011 bis zum Wintersemester 2013/14 im Kurs „Interdisziplinäre Onkologie“ der Trainingsfall „60jährige Patientin mit bioptisch gesichertem Enchondrom“ angeboten, der neben sechs Multiple-Choice-Fragen auch drei Bildmarkierungsaufgaben enthält. Bei jeder dieser Aufgaben muss eine einzelne Region markiert werden (Aufgabe A: endochromatöse Läsion, Aufgabe B: tumoröse Läsion, Aufgabe C: tumoröses Gewebe).

Der Trainingsfall wurde von 164 verschiedenen Studierenden insgesamt 206 mal gestartet und 192 mal vollständig bearbeitet. Eine vollständige Bearbeitung bezeichnet die Bearbeitung aller Fragen bzw. Aufgaben und die Betrachtung der Auswertung des Trainingsfalls. Die Bildmarkierungsaufgaben wurden dabei 200 mal (Aufgabe A) bzw. 192 mal (Aufgabe B und C) bearbeitet, bei einer durchschnittlichen Bearbeitungsdauer von 58, 35 und 40 Sekunden.[2] Die Bearbeitungsdauer schließt das Lesen des Fragetextes mit ein, wird also vom Zeitpunkt der initialen Anzeige von Fragetext und Bild bis zum

[2]Die Erfassung der Bearbeitungsdauer wurde im CaseTrain-System erst im Laufe der Fallstudie eingeführt, so dass sie nicht für alle Bearbeitungen, sondern nur für 116 (Aufgabe A) bzw. 117 (Aufgabe B und C) Bearbeitungen vorliegt.

Bestätigen der Markierungen gemessen. Bei den Multiple-Choice-Fragen lag die durchschnittliche Bearbeitungszeit bei 55 Sekunden (bei insgesamt 700 Bearbeitungen).

Die durchschnittliche Bewertung jeder Bildmarkierungsaufgabe betrug 31 %, 64 % und 78 %, was einem gesamten Durchschnitt von etwa 57 % entspricht. Die durchschnittliche Bewertung der Multiple-Choice-Fragen war etwa 78 %. Die ausführliche Erklärung zu Aufgabe A wurde 125 mal aufgerufen und nach durchschnittlich 24,5 Sekunden wieder geschlossen, bei Aufgabe B waren es 89 Aufrufe und 13,1 Sekunden, bei Aufgabe C 55 Aufrufe und 12,4 Sekunden.

	Anzahl Bearbeitungen	Ø Bearbeitungsdauer/s	Ø Bewertung	Anzahl Aufrufe Erklärung	Ø Betrachtungsdauer/s Erklärung
Aufgabe A	200	58	31 %	125	24,5
Aufgabe B	192	35	65 %	89	13,1
Aufgabe C	192	40	78 %	55	12,4

Tabelle 5.1.: Bearbeitungsstatistiken der Bildmarkierungsaufgaben im Trainingsfall „60-jährige Patientin mit bioptisch gesichertem Enchondrom“

Im Anschluss an die Fallbearbeitung wurde direkt innerhalb des CaseTrain-Players eine Nutzerbefragung durchgeführt, die unter anderem drei Single-Choice-Fragen enthielt. Abbildungen 5.8 bis 5.10 zeigen die Antworten auf diese Fragen. n steht dabei für die Anzahl der Antworten, mw für den Mittelwert, σ für die Standardabweichung und md für den Median. Die Berechnung dieser Werte basiert darauf, den Antwortmöglichkeiten numerische Werte von 1 (trifft voll und ganz zu) bis 5 (trifft überhaupt nicht zu) zuzuordnen.

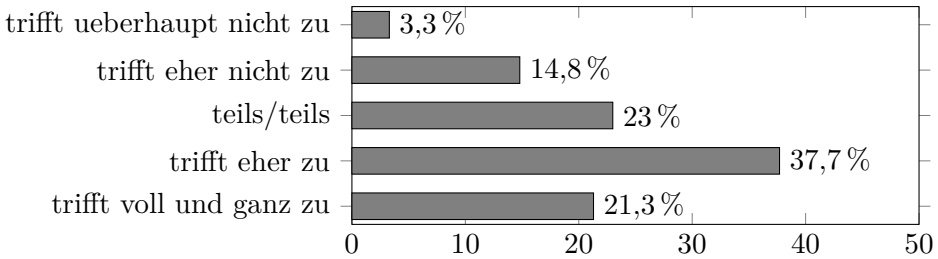


Abbildung 5.8.: Antworten auf die Frage: „Bitte bewerten Sie folgende Aussage: Das Zeichnen der Markierungen mit dem Freihand-Zeichentool ist einfach und intuitiv.“ ($n = 61$, $mw \approx 2{,}4$, $\sigma \approx 1{,}1$, $md = 2$)

Es bestand ebenso die Möglichkeit, Kritik in Form von Freitext zu äußern. Einige Studierende schlugen hier Verbesserungen für das Zeichenwerkzeug vor. Es handelt sich dabei allerdings nur um wenige ($n = 18$), schwierig klassifizierbare Antworten bzw. Vorschläge, so dass von einer systematischen Auswertung abgesehen wird.

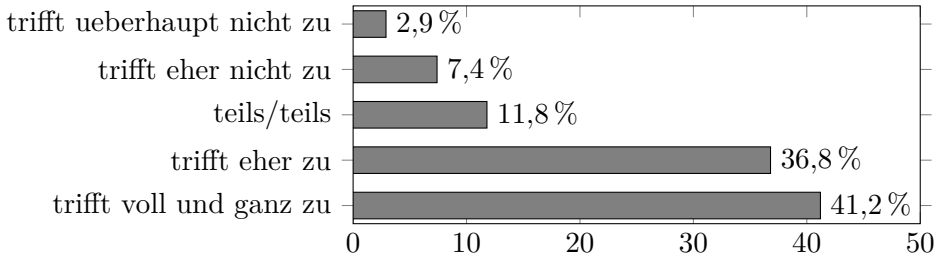


Abbildung 5.9.: Antworten auf die Frage: „Bitte bewerten Sie folgende Aussage: Durch die graphische Darstellung von Musterlösung und eigener Markierung wurde gut vermittelt, welche Fehler Sie gemacht haben.“ ($n = 68$, $mw \approx 1{,}9$, $\sigma \approx 1{,}0$, $md = 2$))

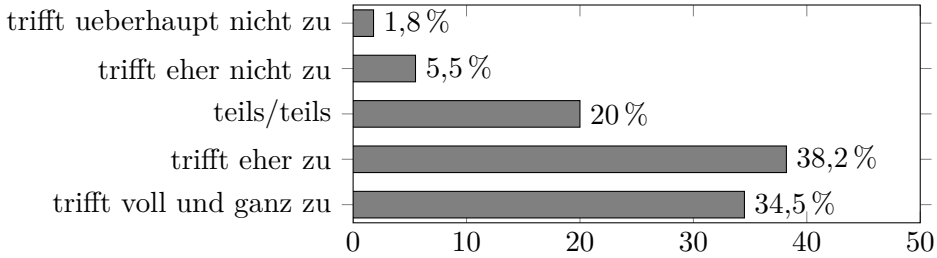


Abbildung 5.10.: Antworten auf die Frage: „Bitte bewerten Sie folgende Aussage: Durch die textuelle Erklärung bei den Markierungs-Fragen wurde gut vermittelt, wie die Bewertung Ihrer Lösung zustande kam.“ ($n = 55$, $mw \approx 1{,}9$, $\sigma \approx 0{,}9$, $md = 2$))

5.2.5. Diskussion und Ausblick

Das Experiment hat gezeigt, dass das Freihand-Zeichenwerkzeug für Studierende (der Medizin) grundsätzlich bedienbar, also effektiv ist. Ursächlich für Schwierigkeiten bei der Bedienung (siehe Abbildung 5.8) könnten unter anderem gewesen sein, dass für das Werkzeug keinerlei Einführungen oder Hilfefunktionen zur Verfügung standen, so dass die Studierenden bei diesem Trainingsfall zum ersten mal damit konfrontiert wurden. Wie die Benutzerumfrage zeigt, schätzte in der Tat knapp ein Fünftel der Studierenden die Bedienung eher nicht als einfach und intuitiv ein (siehe Abbildung 5.8). Hier sollte also noch mehr Wert auf die Gebrauchstauglichkeit des Zeichenwerkzeugs gelegt werden. Konkret erscheint es sinnvoll, das Werkzeug so zu erweitern, dass Markierungen nachträglich einfach modifiziert oder mehrere Markierungen zu einer einzelnen zusammengefügt werden können.

Auffällig ist, dass die Bewertung der ersten Bildmarkierungsaufgabe mit durchschnittlich 31 % trotz gleicher Schwierigkeit deutlich schlechter ausfiel als die beiden folgenden Aufgaben (65 % und 78 %). Eine Ursache könnte sein, dass den Lernenden die Funktionsweise des Werkzeugs bzw. des Auswertungsverfahrens zu Beginn nicht klar war. Dafür sprechen auch die Zahlen zur durchschnittlichen Bearbeitungsdauer, welche bei

der ersten Frage mit 58 Sekunden noch relativ hoch ist, sich bei den weiteren Fragen aber auf 35 bzw. 40 Sekunden reduziert.

Es zeigte sich, dass die Studierenden das Werkzeug nach kurzer Eingewöhnung offenbar ausreichend gut bedienen können. Dies belegen sowohl die geringeren Bearbeitungsdauern der Fragen B und C, als auch die relativ hohen durchschnittlichen Bewertungen, die jeweils ähnlich den Durchschnittswerten der Multiple-Choice-Fragen sind.

Des Weiteren auffällig ist die sinkende Nutzung der ausführlichen Erklärung. Sowohl die Anzahl der Aufrufe, als auch die Betrachtungsdauer gehen bei der zweiten und dritten Aufgabe deutlich zurück. Es ist anzunehmen, dass den Lernenden nach einer erstmaligen ausführlichen Erklärung die Funktionsweise klar geworden ist, zumal die korrekten Markierungen auch ohne Aufrufen der ausführliche Erklärung einsehbar sind (siehe Abbildung 5.6). Eine weitere mögliche Ursache für die sinkende Nutzung der ausführlichen Erklärung ist, dass bei Aufgabenbearbeitungen mit guten Bewertungen die Nutzung der ausführlichen Erklärung erfahrungsgemäß niedriger ist als bei Bearbeitungen mit schlechten Bewertungen. Grundsätzlich sind die Studierenden mit den grafischen und verbalen Erklärungen zufrieden (siehe Abbildung 5.9 und 5.10).

Verbesserungsmöglichkeiten gibt es noch bei der Berechnung der Bewertung der Lernerlösung. Aktuell wird eine Lösung nicht schlechter bewertet, wenn sie neben korrekt gezeichneten noch weitere, überflüssige Polygone enthält. Hier könnten von der Gesamtbewertung noch Prozentpunkte abgezogen werden, beispielsweise in Abhängigkeit von Anzahl und Fläche der überflüssigen Polygone.

Wir haben mit Bildmarkierungsaufgaben einen Aufgabentyp konzipiert und für das CaseTrain-System implementiert, der sich näher an beruflicher, praktischer Anwendung orientiert als die in Trainingssystemen üblicherweise verwendeten geschlossenen Aufgaben. Die Bearbeitung erfordert kognitive Prozesse, die in Blooms Lernzieltaxonomie in der Stufe der *Anwendung* anzusiedeln sind. Auch hier gilt aber die Einschränkung auf das ultimative Dilemma beim Messen kognitiver Prozesse [Haladyna, 2004]: Wenn Studierende bereits ähnliche Fälle bzw. Bilder kennen, werden auch hier nur kognitive Prozesse der Stufe *Wissen* abgerufen.

Leider lässt sich bei dem verwendeten Editor nicht ausschließen, dass teilweise auch Prozesse gefordert und bewertet werden, die möglicherweise nicht relevant für das Lernziel sind. In diesem Fall ist es die technische Auswahl der Regionen durch Mausgesten mit dem Freihandzeichen-Werkzeug. Studierende, die damit handwerklich Probleme haben, werden schlechter bewertet, obwohl sie möglicherweise gleiche fachliche Kompetenz besitzen.

Der Editor wurde mittlerweile um eine Funktionalität erweitert, mit der Markierungen durch die explizite Eingabe von Eckpunkten eines Polygons eingegeben werden können. Diese Eckpunkte können auch nach dem Erstellen der Markierung noch verschoben oder gelöscht werden. Auch können neue Eckpunkte erstellt werden, so dass nun ein unkompliziertes Erstellen von Markierungen möglich ist. Ein nächster Schritt ist die Evaluierung des Trainingssystems mit dem modifizierten Editor.

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde nach einer kurzen Einführung in die Thematik des E-Learning betrachtet, wie Übungsaufgaben klassifiziert werden können. Als Erweiterung der aus der Literatur bekannten Klassifikationen nach benötigten kognitiven Anforderungen und der Form der Aufgabe, wurde eine Klassifikation nach Strukturierbarkeit und maschineller Interpretierbarkeit der Lösungen vorgestellt. Die daraus abgeleiteten Eigenschaften können ausgenutzt werden, um elektronische Trainingssysteme zu konzipieren, die auch Lösungen für offene Aufgaben auswerten und entsprechendes Feedback generieren können.

Lösungen zu strukturierten Aufgaben können mittels des Teile-und-Herrsche-Ansatzes aufgespalten werden, wobei die Lernerlösungen mit einer oder mehreren Musterlösungen verglichen werden können. Dabei kann die Berechnung des allgemein definierten Überdeckungsmaßes verwendet werden, wobei Feedback der Formen *Knowledge of the correct answer* (KCR, Anzeige einer Musterlösung), *Knowledge of performance* (KP, quantitative Bewertung) und *Elaborated feedback* (EF, Erläuterungen der Auswertung) generiert wird. Auch mit regelbasierten Ansätzen lässt sich bei strukturierten Aufgaben auf vielfältige Weise Feedback erzeugen. Beispielsweise oberflächliche Eigenschaften wie der Stil, die Einhaltung von Konventionen, aber auch die semantische Korrektheit der Lösung lassen sich damit prüfen.

Maschinell interpretierbare Aufgaben, die eine Unterklasse der strukturierten Aufgaben darstellen, können von Trainingssystemen direkt ausgeführt werden. Hier lässt sich ein Ergebnisvergleich durchführen, wobei das genaue Verfahren davon abhängt, ob es sich um datenverarbeitende Aufgaben handelt und dazu entweder Eingabedaten und zugehörige korrekte Ausgabedaten, oder Eingabedaten und Musterlösungen vorliegen. Handelt es sich nicht um datenverarbeitende Aufgaben, muss entweder ein korrektes Ergebnis in statischer Form oder eine Musterlösung vorliegen. Das Ergebnis der Ausführung der Lernerlösung kann dann zur Feedback-Generierung mit dem korrekten Ergebnis oder dem Ergebnis der Ausführung der Musterlösung verglichen werden.

Im Hauptteil der Arbeit wurden vier Trainingssysteme bzw. deren Komponenten beschrieben und es wird von den Erfahrungen mit deren Einsatz in der Praxis berichtet. Dabei ergab sich beim Trainingssystem zu UML Klassendiagrammen, dass tatsächlich ein Ansatz nötig ist, der in gewissem Maße tolerant beim Vergleich mit Musterlösungen ist, da ein Großteil der von Studierenden erstellten Klassendiagramme Elemente aufweist, die gegenüber den Elementen aus der Musterlösung unterschiedlich benannt, aber dennoch korrekt oder teilweise korrekt sind. Die Toleranz des Ansatzes führte dabei zu wesentlichen Verbesserungen gegenüber einem ansonsten analogen Ansatz, der

nur solche Elemente der Lernerlösungen akzeptiert, die identisch zu Elementen aus der Musterlösung sind.
Der Ansatz bietet damit einen Mehrwert gegenüber den in der Literatur beschriebenen Trainingssystemen, die diesbezüglich nicht tolerant sind.
Die Aufgaben wurden in ein geführtes Tutorsystem mit zunehmender Selbstständigkeit nach dem Prinzip des *Cognitive Apprenticeship Models* als Komponente des fallbasierten Trainingssystems CaseTrain integriert und zwischen 2011 und 2014 in vier Veranstaltungen im Übungsbetrieb und zum freien Üben produktiv eingesetzt und evaluiert.
Der Aufwand für Dozenten, Aufgaben mit diesem Trainingssystem zu stellen ist sehr gering, da neben einer Aufgabenbeschreibung nur eine Musterlösung hinterlegt werden muss. Auch stieß dessen Einsatz bei den Studierenden auf breite Akzeptanz: 90 % der Befragten hielten einen weiteren Einsatz des Systems für sinnvoll.
Dennoch lassen sich zahlreiche Verbesserungsmöglichkeiten aufzeigen, deren Implementierung das nächste Ziel darstellt. Im besonderen sind hier regelbasierte Möglichkeiten der Überprüfung zu nennen, mit der Klassendiagramme beispielsweise auf die Einhaltung bestimmter Konventionen überprüft werden können. Auch sollten überzählige Assoziationen und Vererbungsbeziehungen korrekt angegebener Klassen negativ bewertet werden, wozu eine Erweiterung des Trainingssystems nötig ist.
Vielversprechend erscheint auch eine Erweiterung, die den Rückfluss von Annotationen bei manueller Nachkorrektur durch Lehrpersonal ausnutzt und so die Zuordnungsprobleme bei der Berechnung des Überdeckungsmaßes besser lösen kann. Dieser Ansatz lässt sich auch auf andere Trainingssysteme übertragen, die das Überdeckungsmaß zur Feedback-Generierung nutzen, beispielsweise auf ein Trainingssystem, das Lösungsskizzen automatisch auswertet.

Das neuartige Trainingssystem „WARP“, in dem als Lösungen UML Aktivitätsdiagramme eingegeben werden, nutzt die starke Strukturierung und die durch die Übersetzbarkeit in Programmcode entstehende maschinelle Interpretierbarkeit aus, um Feedback in mehreren Ebenen zu generieren. Die Aufgaben werden dabei im Kontext einer Roboter-Simulation-Umgebung gestellt, wobei die Lösungen der Lernenden das Verhalten eines Roboters in einem bestimmten Szenario definieren. In einer durch das Szenario festgelegten virtuellen Umgebung muss der Roboter zum Bestehen der Aufgabe ein bestimmtes Ziel erreichen. Feedback erhalten Lernende hier vor allem in Form einer Animation, welche ihnen als Rückmeldung zum Verhalten ihres modellierten Roboters dient.
WARP wurde 2013 und 2014 im Rahmen zweier Veranstaltungen im Übungsbetrieb und zum freien Üben produktiv eingesetzt und evaluiert. Die Szenarien wurden im zweiten Experiment durch stochastische Elemente und ein kompetitives Mehrspieler-Szenario erweitert, in welchem die Lernenden gegen einen vom System definierten Standardgegner, oder zum freien Üben auch gegeneinander antreten konnten.
Trotz einiger, teils schwerwiegender technischer Probleme wurde WARP von den Studierenden zufriedenstellend, das Konzept unabhängig von der Implementierung sogar gut bewertet. Zudem befürworteten 78 % der Studierenden einen weiteren Einsatz von WARP. Auch hier bietet sich noch viel Raum für Verbesserungen, nicht nur technischer, sondern auch konzeptioneller Art. Herauszuheben ist hier die Implementierung einer wei-

teren Feedback-Schleife, in der Aktivitätsdiagramme mittels regelbasierter Verfahren auf „Diagram Smells“ und auf die Einhaltung bestimmter Konventionen untersucht werden.

Im Trainingssystem „ÜPS“ müssen zum Lösen von Aufgaben SQL Anfragen eingegeben werden. Diese SQL Anfragen werden nach einer syntaktischen Überprüfung und einem statischen Vergleich mit einer Musterlösung im Kontext einer gefüllten Datenbank ausgeführt. Die Beschreibung des zu einer Datenbank gehörenden Szenarios bietet den Lernenden einen narrativen Anker und soll deren Aufmerksamkeit auf die Problemstellung lenken. Nach der Ausführung der Lernerlösung wird auch eine Musterlösungs-Anfrage ausgeführt, wobei ein automatischer Vergleich der entsprechenden Ergebnisse stattfindet, aus welchem eine Rückmeldung generiert wird.
ÜPS wurde 2013 und 2014 im Übungsbetrieb und zum freien Üben in zwei Vorlesungen produktiv eingesetzt, wobei den Studierenden Aufgaben in ein bzw. zwei Szenarien angeboten wurden. Die Möglichkeit zum freien Üben wurde von den Studierenden sehr gut angenommen. Umfragen ergaben auch hier eine in allen Aspekten gute Bewertung und eine breite Akzeptanz des Trainingssystems, dessen Einsatz von 88 % der befragten Studierenden befürwortet wird.
Den zuvor in dieser Arbeit vorgestellten Trainingssystemen hat ÜPS voraus, dass Lehrpersonal noch während des laufenden Übungsbetriebs in die automatische Bewertung eingreifen kann. So kann beispielsweise frühzeitig erkannt werden, wenn fehlerhafte Musterlösungen hinterlegt worden sind, was dadurch auffällt, dass die vorläufigen Bewertungen für die entsprechende Aufgabe wesentlich schlechter ausfallen als andere. Die Musterlösungen können modifiziert werden, was eine unmittelbare Neubewertung bisher eingereichter Lösungen folgen lässt.
Ein Vorteil von ÜPS gegenüber den in der Literatur vorkommenden Trainingssystemen ist die starke Trennung von technisch-administrativen einerseits und inhaltlichen Aufgaben andererseits. Dieses Konzept sollte in einer Erweiterung dazu genutzt werden, auch Lernenden die Erstellung von Szenarien und Aufgaben zu ermöglichen, was eine neue Aufgabenform mit höheren kognitiven Anforderungen darstellt und gleichzeitig für mehr Übungsmöglichkeiten für Kommilitonen sorgt.

Die in CaseTrain implementierte Komponente für Bildmarkierungsaufgaben ermöglicht eine unmittelbare, automatische Bewertung entsprechender Aufgaben. Sie ist ein Beleg für die Anwendbarkeit der in Abschnitt 1.6 vorgestellten Konzepte zur Feedback-Generierung strukturierter Aufgaben auch in von der Informatik verschiedenen Fachgebieten, beispielsweise in der Medizin. Das Trainingssystem wurde zwischen 2011 und 2014 in einer Veranstaltung zum freien Üben produktiv eingesetzt und evaluiert. Auch dieses Trainingssystem wurde von den Studierenden gut angenommen, wobei es noch Verbesserungsbedarf bei der Gebrauchstauglichkeit des eingesetzten Editors gab. Dieser wurde inzwischen überarbeitet, so dass eine erneute Evaluation stattfinden sollte.

In den Fallstudien wurde keine systematische Evaluierung des Lernerfolgs, beispielsweise anhand von Klausurergebnissen, vorgenommen. Dies hatte vor allem pragmatische Gründe: Die Experimente wurden größtenteils im Übungsbetrieb der entsprechenden Vorlesungen durchgeführt, der für die Studierenden wegen der nur durch die erfolgreiche Teilnahme erhältlichen Zulassung für die Klausur auch hohe formale Wichtigkeit hat.

Daher sollten für die Studierenden möglichst gleiche Bedingungen während des Übungsbetriebs herrschen, so dass von Evaluierungen durch Bildung von Parallelgruppen und Vergleich deren Klausurergebnisse abgesehen wurde. Auch die Anzahl der Studierenden, die freiwillig nicht an einzelnen Aufgaben im Übungsbetrieb teilgenommen hatten, war zu klein, um belastbare Aussagen über den Lernerfolg treffen zu können.

Abschließend lässt sich sagen, dass sich für Trainingssysteme, die offene Aufgaben durch das Ausnutzen der Eigenschaften der starken Strukturierung und der maschinellen Interpretierbarkeit maschinell auswerten können, vor allem in naturwissenschaftlichen, aber auch anderen Fachgebieten, wohl noch viele weitere Einsatzmöglichkeiten finden lassen. Studierende befürworten den Einsatz elektronischer Trainingssysteme zur Feedback-Generierung. Dies wird nicht nur durch die Evaluationsergebnisse der hier vorgestellten Trainingssysteme, sondern durch auch die grundsätzliche Befürwortung des Einsatzes elektronischer Trainingssysteme belegt: von 240 befragten Studierenden aus der Informatik und verwandten Studiengängen stimmten 93 % für den Einsatz elektronischer Trainingssysteme. Doch auch über die Grenzen dieses Fachgebiets hinaus scheint die Akzeptanz der Studierenden groß zu sein: Bei den jedes Semester zum Einsatz des fallbasierten Trainingssystem CaseTrain durchgeführten fakultätsübergreifenden Umfragen an der Universität Würzburg sprechen sich die Studierenden seit mehreren Jahren mit Werten zwischen 95 % und 98 % in sehr hohem Maße für Ausweitung des Einsatzes auf weitere Veranstaltungen aus. Da der Aufwand für das Lehrpersonal, strukturierte Aufgaben in entsprechenden Trainingssystemen zu erstellen bei entsprechender Konzeption sehr gering sein kann, sollte in allen Fachgebieten nach Möglichkeiten gesucht werden, den Studierenden weitere praxisnahe Übungsmöglichkeiten zur Verfügung zu stellen.

A. Anhang

A.1. Quellcode

A.1.1. XML

A.1.1.1. XML-Schema für UML-Klassendiagramme

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="UMLClass">
    <xsd:complexContent>
      <xsd:extension base="AbstractLinkable">
        <xsd:sequence maxOccurs="1" minOccurs="1">
          <xsd:element name="title" type="xsd:string" maxOccurs="1"
            minOccurs="1">
          </xsd:element>
          <xsd:element name="attributes" type="TextField"
            maxOccurs="1" minOccurs="1">
          </xsd:element>
          <xsd:element name="functions" type="TextField"
            maxOccurs="1" minOccurs="1">
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="lock_title" type="xsd:boolean"
          use="required"></xsd:attribute>
        <xsd:attribute name="lock_style" type="xsd:boolean"
          use="required"></xsd:attribute>
        <xsd:attribute name="lock_attributes" type="xsd:boolean"
          use="required">
        </xsd:attribute>
        <xsd:attribute name="lock_functions" type="xsd:boolean" />
        <xsd:attribute name="specialType" use="optional">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="abstract" />
              <xsd:enumeration value="interface" />
              <xsd:enumeration value="enumeration" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
```

```
</xsd:complexType>

<xsd:element name="umldiagram" type="UMLDocument"></xsd:element>

<xsd:complexType name="UMLDocument">
  <xsd:all maxOccurs="1" minOccurs="1">
    <xsd:element name="classes" maxOccurs="1" minOccurs="1">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="class" type="UMLClass" maxOccurs="unbounded"
            minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="associations" maxOccurs="1" minOccurs="1">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="association" type="Association"
            maxOccurs="unbounded" minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="derivations" maxOccurs="1" minOccurs="1">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="derivation" type="AbstractLink"
            maxOccurs="unbounded" minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="associationlinks" maxOccurs="1"
      minOccurs="1">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="associationlink" type="AbstractLink"
            maxOccurs="unbounded" minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="nthlinkables" maxOccurs="1" minOccurs="1">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="nth" type="NthLinkable" maxOccurs="unbounded
              "
            minOccurs="0">
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="availableattributes" type="UnboundedStringList"
```

```
      maxOccurs="1" minOccurs="0">
    </xsd:element>
    <xsd:element name="availablefunctions" type="UnboundedStringList"
      maxOccurs="1" minOccurs="0"></xsd:element>
  </xsd:all>
  <xsd:attribute name="width" type="xsd:double" use="required"></
      xsd:attribute>
  <xsd:attribute name="height" type="xsd:double" use="required"></
      xsd:attribute>
</xsd:complexType>

<xsd:complexType name="Association">
  <xsd:complexContent>
    <xsd:extension base="AbstractLink">
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string" maxOccurs="1"
          minOccurs="1">
        </xsd:element>
        <xsd:element name="capA" type="Cap" maxOccurs="1"
          minOccurs="1">
        </xsd:element>
        <xsd:element name="capB" type="Cap"></xsd:element>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:int" use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Point">
  <xsd:attribute name="id" type="xsd:int" use="required" />
  <xsd:attribute name="glue" type="xsd:int" use="required" />
  <xsd:attribute name="x" type="xsd:double" use="required" />
  <xsd:attribute name="y" type="xsd:double" use="required" />
</xsd:complexType>

<xsd:complexType name="TextField">
  <xsd:sequence>
    <xsd:element name="line" type="xsd:string" maxOccurs="unbounded"
      minOccurs="0"></xsd:element>
  </xsd:sequence>
  <xsd:attribute name="expanded" type="xsd:boolean" use="required">
  </xsd:attribute>
</xsd:complexType>



<xsd:complexType name="Cap">
  <xsd:attribute name="style" type="xsd:int" use="required"></
      xsd:attribute>
  <xsd:attribute name="cardinality" type="xsd:string" use="required"></
      xsd:attribute>
</xsd:complexType>
```

```
  <xsd:complexType name="NthLinkable">
    <xsd:complexContent>
      <xsd:extension base="AbstractLinkable">
        <xsd:sequence>
          <xsd:element name="title" type="xsd:string" maxOccurs="1"
            minOccurs="1"></xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="AbstractLink">
    <xsd:sequence>
      <xsd:element name="start" type="Point" maxOccurs="1"
        minOccurs="1">
      </xsd:element>
      <xsd:element name="end" type="Point" maxOccurs="1"
        minOccurs="1">
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="lock_position" type="xsd:boolean"
      use="required" />
    <xsd:attribute name="lock_delete" type="xsd:boolean"
      use="required" />
    <xsd:attribute name="route" type="xsd:int" />
  </xsd:complexType>

  <xsd:complexType name="AbstractLinkable">
    <xsd:attribute name="id" type="xsd:int" use="required"></xsd:attribute>
    <xsd:attribute name="x" type="xsd:double" use="required"></
        xsd:attribute>
    <xsd:attribute name="y" type="xsd:double" use="required"></
        xsd:attribute>
    <xsd:attribute name="lock_position" type="xsd:boolean"
      use="required"></xsd:attribute>
    <xsd:attribute name="lock_delete" type="xsd:boolean"
      use="required"></xsd:attribute>
  </xsd:complexType>

  <xsd:complexType name="UnboundedStringList">
    <xsd:sequence>
      <xsd:element name="element" type="xsd:string" maxOccurs="unbounded"
        minOccurs="0"></xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

A.1.1.2. XML-Repräsenation von UML-Klassendiagrammen

Beispiel Bibliothek (Musterlösung):

```
<umldiagram width="914" height="914">
  <classes>
```

```
<class x="221" y="8" id="1" lock_position="false" lock_delete="false"
    lock_title="false" lock_attributes="false" lock_functions="false"
    lock_style="false">
  <title>Einrichtung</title>
  <attributes expanded="true">
    <line>name : String</line>
  </attributes>
  <functions expanded="true">
    <line></line>
  </functions>
</class>
<class x="206" y="108" id="3" lock_position="false" lock_delete="false"
     lock_title="false" lock_attributes="false" lock_functions="false"
    lock_style="false">
  <title>Bibliothek</title>
  <attributes expanded="true">
    <line></line>
  </attributes>
  <functions expanded="true">
    <line>sperren(Benutzer)</line>
    <line>anlegen(Benutzer)</line>
  </functions>
</class>
<class x="400" y="85" id="5" lock_position="false" lock_delete="false"
    lock_title="false" lock_attributes="false" lock_functions="false"
    lock_style="false">
  <title>Benutzer</title>
  <attributes expanded="true">
    <line>id : long</line>
    <line>vorname : String</line>
    <line>nachname : String</line>
    <line>geburtsdatum : Date</line>
    <line>istDozent : boolean</line>
  </attributes>
  <functions expanded="true">
    <line>leihen(Buchexemplar)</line>
    <line>zurÃŒckgeben(Buchexemplar)</line>
    <line>istGesperrt() : boolean</line>
  </functions>
</class>
<class x="220" y="236" id="10" lock_position="false" lock_delete="false
    " lock_title="false" lock_attributes="false" lock_functions="false"
     lock_style="false">
  <title>Buch</title>
  <attributes expanded="true">
    <line>titel : String</line>
    <line>autor : String</line>
    <line>jahr : short</line>
  </attributes>
  <functions expanded="true">
    <line></line>
  </functions>
</class>
```

```
    <class x="219" y="366" id="12" lock_position="false" lock_delete="false
        " lock_title="false" lock_attributes="false" lock_functions="false"
         lock_style="false">
      <title>Buchexemplar</title>
      <attributes expanded="true">
        <line>signatur : String</line>
      </attributes>
      <functions expanded="true">
        <line></line>
      </functions>
    </class>
  </classes>
  <associations>
    <association lock_position="false" lock_delete="false" route="2" id="8"
        >
      <start id="3" glue="12" x="269" y="265"/>
      <end id="5" glue="12" x="468" y="265"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
    <association lock_position="false" lock_delete="false" route="1" id="14
        ">
      <start id="3" glue="12" x="267" y="253"/>
      <end id="10" glue="12" x="349" y="130"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
    <association lock_position="false" lock_delete="false" route="1" id="16
        ">
      <start id="10" glue="12" x="239.65" y="390"/>
      <end id="12" glue="1" x="240" y="429"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
    <association lock_position="false" lock_delete="false" route="2" id="18
        ">
      <start id="10" glue="12" x="280" y="366"/>
      <end id="5" glue="6" x="442" y="315"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
    <association lock_position="false" lock_delete="false" route="2" id="20
        ">
      <start id="12" glue="12" x="282" y="469"/>
      <end id="5" glue="8" x="499" y="313"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
  </associations>
```

```
  <derivations>
    <derivation lock_position="false" lock_delete="false" route="1">
      <start id="3" glue="12" x="251" y="265"/>
      <end id="1" glue="12" x="242" y="140"/>
    </derivation>
  </derivations>
  <nthlinkables/>
  <associationlinks/>
</umldiagram>
```

Beispiel Bibliothek (Lernerlösung):

```
<umldiagram width="914" height="914">
  <classes>
    <class x="216" y="133" id="3" lock_position="false" lock_delete="false"
         lock_title="false" lock_attributes="false" lock_functions="false"
        lock_style="false">
      <title>Bibliotek</title>
      <attributes expanded="true">
        <line>name : String</line>
      </attributes>
      <functions expanded="true">
        <line>sperren(Person)</line>
      </functions>
    </class>
    <class x="398" y="86" id="5" lock_position="false" lock_delete="false"
        lock_title="false" lock_attributes="false" lock_functions="false"
        lock_style="false">
      <title>Person</title>
      <attributes expanded="true">
        <line>ID : int</line>
        <line>vorname : String</line>
        <line>zuname : String</line>
        <line>istDozent : Boolean</line>
      </attributes>
      <functions expanded="true">
        <line>ausleihen(Buch)</line>
        <line>zurÃŒckgeben(Exemplar)</line>
        <line>gespert() : String</line>
      </functions>
    </class>
    <class x="218" y="265" id="10" lock_position="false" lock_delete="false
        " lock_title="false" lock_attributes="false" lock_functions="false"
         lock_style="false">
      <title>Buch</title>
      <attributes expanded="true">
        <line>titel : String</line>
        <line>author : String</line>
      </attributes>
      <functions expanded="true">
        <line></line>
      </functions>
    </class>
```

```
    <class x="214" y="412" id="12" lock_position="false" lock_delete="false
        " lock_title="false" lock_attributes="false" lock_functions="false"
         lock_style="false">
      <title>Exemplar</title>
      <attributes expanded="true">
        <line>sigantur : int</line>
      </attributes>
      <functions expanded="true">
        <line></line>
      </functions>
    </class>
    <class x="215" y="48" id="22" lock_position="false" lock_delete="false"
         lock_title="false" lock_attributes="false" lock_functions="false"
        lock_style="false">
      <title>Student</title>
      <attributes expanded="true">
        <line></line>
      </attributes>
      <functions expanded="true">
        <line></line>
      </functions>
    </class>
  </classes>
  <associations>
    <association lock_position="false" lock_delete="false" route="2" id="8"
        >
      <start id="3" glue="12" x="269" y="265"/>
      <end id="5" glue="12" x="468" y="265"/>
      <title></title>
      <capA style="0" cardinality="*"/>
      <capB style="0" cardinality="1"/>
    </association>
    <association lock_position="false" lock_delete="false" route="1" id="14
        ">
      <start id="3" glue="12" x="267" y="253"/>
      <end id="10" glue="12" x="349" y="130"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
    <association lock_position="false" lock_delete="false" route="2" id="18
        ">
      <start id="10" glue="12" x="280" y="366"/>
      <end id="5" glue="6" x="442" y="315"/>
      <title></title>
      <capA style="0" cardinality="1"/>
      <capB style="0" cardinality="*"/>
    </association>
    <association lock_position="false" lock_delete="false" route="2" id="20
        ">
      <start id="12" glue="12" x="282" y="469"/>
      <end id="5" glue="8" x="499" y="313"/>
      <title></title>
      <capA style="0" cardinality="*"/>
```

```
      <capB style="0" cardinality="*"/>
    </association>
  </associations>
  <derivations>
    <derivation lock_position="false" lock_delete="false" route="2">
      <start id="12" glue="12" x="266" y="419"/>
      <end id="10" glue="7" x="268" y="342"/>
    </derivation>
    <derivation lock_position="false" lock_delete="false" route="2">
      <start id="22" glue="3" x="316" y="62"/>
      <end id="5" glue="1" x="478" y="86"/>
    </derivation>
  </derivations>
  <nthlinkables/>
  <associationlinks/>
</umldiagram>
```

A.1.1.3. XML-Schema für UML-Aktivitätsdiagramme

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
    www.example.org/activity_diagram"
   xmlns:cad="http://www.example.org/activity_diagram" elementFormDefault="
       qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="parameterType">
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="type" type="xs:string" />
  </xs:complexType>

  <xs:complexType name="statement">
    <xs:sequence>
      <xs:element ref="cad:CodeActionBlock" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="connectionPoint">
    <xs:attribute name="blockId" type="xs:int" />
    <xs:attribute name="gluePointsId" type="xs:int" />
  </xs:complexType>

  <xs:complexType name="block">
    <xs:attribute name="id" type="xs:int" />
    <xs:attribute name="x" type="xs:double" />
    <xs:attribute name="y" type="xs:double" />
  </xs:complexType>


  <xs:complexType name="blockContainer">
    <xs:complexContent>
      <xs:extension base="cad:block">
```

```
        <xs:sequence>
          <xs:element name="Blocks">
            <xs:complexType>
              <xs:choice maxOccurs="unbounded" minOccurs="0">
                <xs:element ref="cad:StartBlock" />
                <xs:element ref="cad:EndBlock" />
                <xs:element ref="cad:CodeActionBlock" />
                <xs:element ref="cad:UnionBlock" />
                <xs:element ref="cad:ConditionalBranchBlock" />
                <xs:element ref="cad:ConditionalBlock" />
                <xs:element ref="cad:ForDoBlock" />
                <xs:element ref="cad:WhileDoBlock" />
                <xs:element ref="cad:DoWhileBlock" />
              </xs:choice>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <attribute name="userHeight" type="double"></attribute>
        <attribute name="userWidth" type="double"></attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="ActivityDiagram">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="MetaData">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Author" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="ID" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="ID" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Activity" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="cad:blockContainer">
                <xs:sequence>
                  <xs:element name="Signature">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="Modifiers">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="Modifier" type="xs:string"
                                minOccurs="0" maxOccurs="unbounded" />
                            </xs:sequence>
```

```
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="Name" type="xs:string" />
                  <xs:element name="InputParameters">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="Parameter" type="
                            cad:parameterType"
                          minOccurs="0" maxOccurs="unbounded" />
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="OutputParameter" type="
                      cad:parameterType"
                    minOccurs="0" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="Connection" maxOccurs="unbounded"
    minOccurs="0" type="cad:Connection">
  </xs:element>
 </xs:sequence>
 <attribute name="userHeight" type="double"></attribute>
</xs:complexType>
</xs:element>

<xs:element name="CodeLine" type="xs:string" />

<xs:element name="StartBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block" />
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="EndBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block" />
    </xs:complexContent>
  </xs:complexType>
</xs:element>


<xs:element name="CodeActionBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block">
```

```
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="cad:CodeLine" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="ConditionalBranchBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block">
        <xs:sequence>
          <xs:element ref="cad:CodeLine" />
        </xs:sequence>
        <attribute name="truePos" type="int"></attribute>
        <attribute name="falsePos" type="int"></attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="UnionBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block" />
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="ThenBlock" type="cad:blockContainer" />

<xs:element name="ConditionalBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block">
        <xs:sequence>
          <xs:element name="IfThenComponent">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="IfStatement" type="cad:statement" />
                <xs:element ref="cad:ThenBlock" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>

          <xs:element name="ElseIfThenComponent" minOccurs="0"
            maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ElseIfStatement" type="cad:statement" /
                    >
                <xs:element ref="cad:ThenBlock" />
              </xs:sequence>
```

```
            </xs:complexType>
          </xs:element>

          <xs:element name="ElseBlock" type="cad:blockContainer"
              minOccurs="0" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="WhileStatement" type="cad:statement" />

<xs:element name="ForStatement" type="cad:statement" />

<xs:element name="ForEachStatement" type="cad:statement" />

<xs:element name="DoBlock" type="cad:blockContainer" />

<xs:element name="ForDoBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block">
        <xs:sequence>
          <xs:element ref="cad:ForStatement" />
          <xs:element ref="cad:DoBlock" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="WhileDoBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block">
        <xs:sequence>
          <xs:element ref="cad:WhileStatement" />
          <xs:element ref="cad:DoBlock" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="DoWhileBlock">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="cad:block">
        <xs:sequence>
          <xs:element ref="cad:DoBlock" />
          <xs:element ref="cad:WhileStatement" />
        </xs:sequence>
      </xs:extension>
```

```
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="Connection">
    <xs:sequence>
      <xs:element name="Tail" type="cad:connectionPoint" />
      <xs:element name="Head" type="cad:connectionPoint" />
    </xs:sequence>
    <xs:attribute name="route" type="int" />
  </xs:complexType>
</schema>
```

A.1.1.4. XML-Repräsenation von UML-Aktivitätsdiagramme

Beispiel: Berechnung der Fakultät einer natürlichen Zahl:

```
<ActivityDiagram>
  <MetaData>
    <Author><![CDATA[Marianus und Katharina und Kilian Ifland]]></Author>
    <ID><![CDATA[05091995020720110904 2014]]></ID>
  </MetaData>
  <Activity userHeight="0" id="24" x="0" y="5">
    <Blocks>
      <CodeActionBlock id="26" x="134" y="42">
        <CodeLine><![CDATA[int res;]]></CodeLine>
      </CodeActionBlock>
      <ConditionalBranchBlock truePos="2" falsePos="3" id="29" x="238" y="
          37">
        <CodeLine><![CDATA[n > 0]]></CodeLine>
      </ConditionalBranchBlock>
      <CodeActionBlock id="31" x="239" y="229">
        <CodeLine><![CDATA[res = 1;]]></CodeLine>
      </CodeActionBlock>
      <UnionBlock id="36" x="435" y="227"/>
      <ConditionalBranchBlock truePos="2" falsePos="3" id="45" x="493" y="
          65">
        <CodeLine><![CDATA[n > 1]]></CodeLine>
      </ConditionalBranchBlock>
      <CodeActionBlock id="47" x="647" y="74.5">
        <CodeLine><![CDATA[res = res * n;]]></CodeLine>
      </CodeActionBlock>
      <CodeActionBlock id="49" x="682" y="135.5">
        <CodeLine><![CDATA[n--;]]></CodeLine>
      </CodeActionBlock>
      <StartBlock id="51"/>
      <EndBlock id="52"/>
    </Blocks>
    <Signature>
      <Modifiers/>
      <Name>faktorial</Name>
      <InputParameters>
        <Parameter name="n" type="int"/>
      </InputParameters>
```

```
      <OutputParameter name="res" type="int"/>
    </Signature>
  </Activity>
  <Connection route="2">
    <Tail blockId="51" gluePointsId="0"/>
    <Head blockId="26" gluePointsId="3"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="26" gluePointsId="1"/>
    <Head blockId="29" gluePointsId="0"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="29" gluePointsId="3"/>
    <Head blockId="31" gluePointsId="0"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="31" gluePointsId="1"/>
    <Head blockId="36" gluePointsId="0"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="29" gluePointsId="2"/>
    <Head blockId="45" gluePointsId="4"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="45" gluePointsId="2"/>
    <Head blockId="47" gluePointsId="3"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="47" gluePointsId="2"/>
    <Head blockId="49" gluePointsId="0"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="49" gluePointsId="3"/>
    <Head blockId="45" gluePointsId="3"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="45" gluePointsId="0"/>
    <Head blockId="36" gluePointsId="1"/>
  </Connection>
  <Connection route="2">
    <Tail blockId="36" gluePointsId="2"/>
    <Head blockId="52" gluePointsId="1"/>
  </Connection>
</ActivityDiagram>
```

A.1.2. Java-Code

A.1.2.1. RoSE - Governor-Klasse des Szenarions Labyrinth

```
package rose.governors;

import rose.world.*;
```

```
import rose.robot.Robot;
import rose.robot.RobotState;
import rose.grading.*;
import rose.importer.ServerOptions;

import java.util.*;

public class StopOnRedField extends Governor {
    private final World world;
    private final GradeCollector collector;
    private int turns = 0;
    private boolean onRedField = false;
    boolean isRandomized = false;

    public StopOnRedField(final World world, ServerOptions so) {
        this(world, null, so);
    }

    public StopOnRedField(final World world, final GradeCollector collector
        , ServerOptions so) {
        super(world, collector);
        this.collector = collector;
        this.world = world;
        this.turns = 0;
        if(so.argument != null && so.argument.contains("random"))
            isRandomized = true;
    }

    @Override
    public void onNewRobot(Robot robot) {
        if(isRandomized){
            Node newStart = getValidStartPos();

            //set the Robot to the new position
            RobotState rs = robot.getState();
            RobotState newState = new RobotState(rs.getDirection(),
                newStart);

            robot.setState(newState);
        }

    }

    private Node getValidStartPos(){
        java.util.Random rndm = new Random();
        WorldState ws = world.getWorldState();

        int x = 0;
        int y = 0;

        Node newStart = null;
        boolean isARedField = true;

        //find a node that is not the treasure/red field
```

```
        while(isARedField){
            x = rndm.nextInt(ws.getWidth());
            y = rndm.nextInt(ws.getHeight());

            newStart = new Node(x,y);

            isARedField = this.world.getWorldState()
                    .getColor(newStart).equals(Color.RED);
        }

        return newStart;
    }

    @Override
    public void onStartOfRound() {
        this.turns++;
        if (!this.world.getRobots().isEmpty()) {
            final Node robotPostion = this.world.getRobots().get(0).
                getState().getNode();
            this.onRedField = this.world.getWorldState().getColor(
                robotPostion).equals(Color.RED);
        }

    }

    @Override
    public void onEnd() {
        collector.set(false, -1, -1, "by stopOnRedField", turns);
        collector.addSingleRobotGrade(new SingleRobotGrade(this.onRedField,
             -1, ""));
        if (this.onRedField) {
            this.world.putBroadcast("Labyrinth", "Gewonnen!");
        } else {
            this.world.putBroadcast("Labyrinth", "Verloren!");
        }
    }
}
```

A.2. Übungsbetrieb Vorlesung Softwaretechnik

A.2.1. UML Klassendiagramme: Domänen

A.2.1.1. Universitätsbibliothek

Modellieren Sie eine Bibliothek mit mehreren Standorten. Sie enthält Bücher (Autor, Titel, Verlag, allgemeine Signatur usw.). Zu einem Buch kann es mehrere Buchexemplare (Signatur mit Exemplarkennzeichen) geben, die sich an bestimmten Standorten befinden und teilweise ausleihbar und teilweise Präsenzexemplare sind. Ein Leser (Name, Anschrift usw.) kann Bücher in Katalogen suchen, je nach Status (Student, Dozent) Buchexemplare unterschiedlich lange ausleihen, sie zurückgeben und Bücher vorbestellen. Bei Überschreiten der Ausleihfrist wird er bis zu 3 Mal gemahnt und muss abhängig

vom Überschreiten der Ausleihdauer eine Strafgebühr zahlen. Bei Nichtbezahlen wird er ab einem Gesamtbetrag von x Euro gesperrt. Für jedes Jahr will die Bibliothek eine Statistik aller ausgeliehenen und vorbestellten Bücher haben.

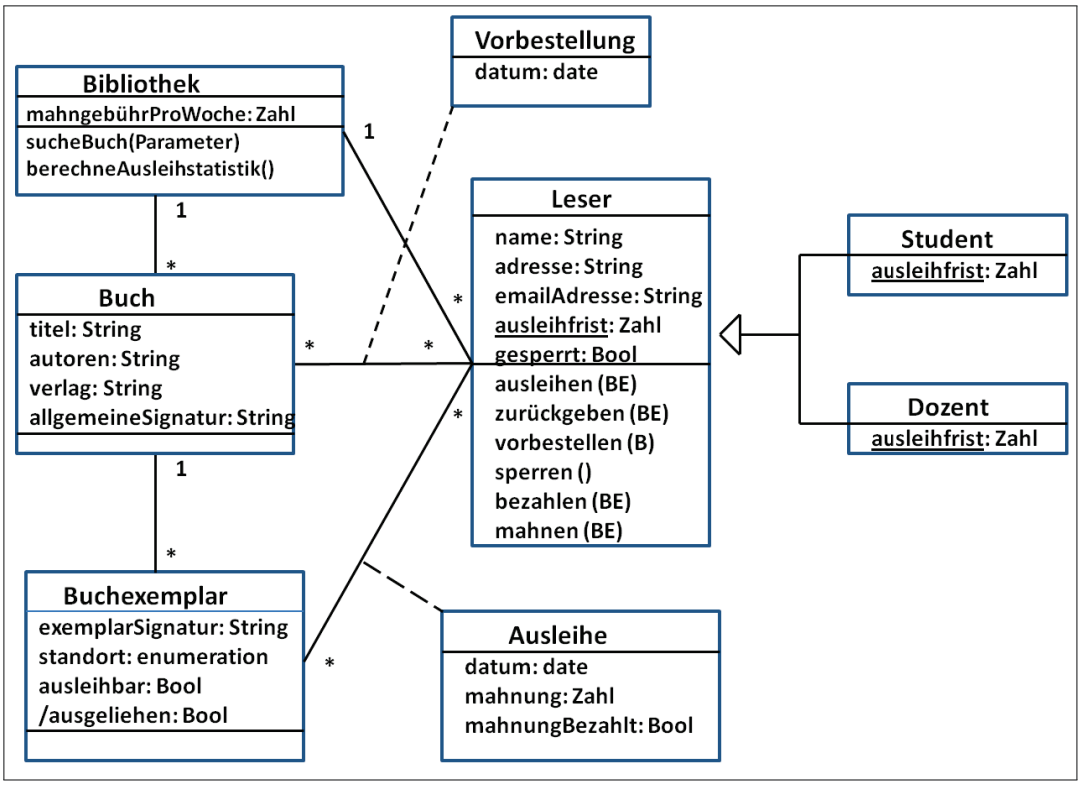


Abbildung A.1.: Musterlösung der Aufgabe „Universitätsbibliothek“

A.2.1.2. Fußball Bundesliga

Es gibt 18 Mannschaften. Diese haben einen Mannschaftsnamen, einen Punktestand und eine Tordifferenz und sind entsprechend in der Tabelle sortiert (pro gewonnenem Spiel gibt es 3 Punkte, bei Unentschieden 1 Punkt und bei Verloren 0 Punkte; die Tordifferenz ist die Differenz der selbst geschossenen Tore und der Gegentore; der Einfachheit halber wird die eigentlich notwendige Zahl der geschossenen Tore nicht bei der Tabellensortierung berücksichtigt). Zu einer Mannschaft gehören Spieler und zu einem Zeitpunkt genau ein Trainer, aber es sollen auch frühere Trainer erfasst werden (nicht aber frühere Spieler). Jede Mannschaft bestreitet gegen jede andere Mannschaft 2 Spiele (ein Heim- und ein Auswärtsspiel). Ein Spiel hat außerdem ein Datum und ein Ergebnis. Weiterhin sollen die Torschützen des Spiels mit Anzahl der Tore, die sie in diesem Spiel geschossen haben, erfasst werden. Bei den Spielern sollen Name, Vorname, Alter, die Gesamtzahl geschossener Tore erfasst werden. Bei Trainern soll Name, Vorname, Alter und auch die früheren Mannschaften erfasst werden, jeweils mit Zeitraum, die der Trainer trainiert hat

(letzteres nur beim Trainer, nicht bei den Spielern). Schließlich soll der Torschützenkönig berechnet werden.

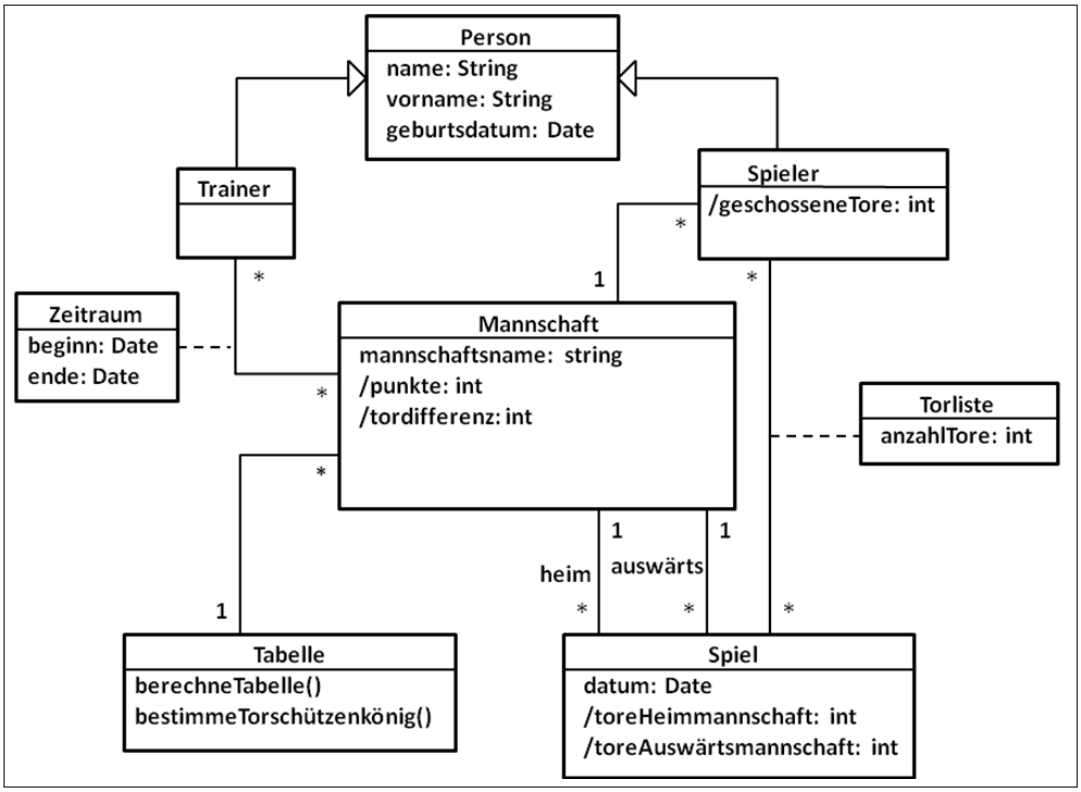


Abbildung A.2.: Musterlösung der Aufgabe „Fußball Bundesliga“

A.2.1.3. Schachpartie

Bei einer Schachpartie treten 2 Spieler an einem Brett mit 64 Feldern gegeneinander an. Jeder Spieler besitzt eine Reihe von Figuren, die unterschiedliche Bewegungen durchführen können. Eine Figur ist entweder ein König, eine Dame, ein Turm, ein Springer, ein Läufer oder ein Bauer. Während der Partie befindet sich jede Figur, die noch nicht geschlagen wurde, auf einem der 64 Felder. Zu Beginn der Partie wird auf dem Spielfeld eine feste Startaufstellung hergestellt. Anschließend ziehen die Spieler abwechselnd. Der aktive Spieler kann einen Zug in das System eingeben, in dem er ein Startfeld und ein Zielfeld angibt. Das System prüft darauf hin, ob sich auf dem Startfeld eine Figur des ziehenden Spielers befindet und die dort befindliche Figur auf das Zielfeld gezogen werden darf (für diese Prüfung dürfen Sie eine Operation "prüfeZug ()"verwenden). Wenn es sich um einen illegalen Zug handelt, so wird vom Spieler die Eingabe eines neuen Zuges verlangt, ansonsten kann der Zug durchgeführt werden. Anschließend prüft das System, ob sich die Partie in einer Schachmattsituation oder Remissituation (z.B. Patt) befindet. Falls nicht, so ist der Gegenspieler an der Reihe. Die Züge sollen wie bei Schachpartien üblich protokolliert werden

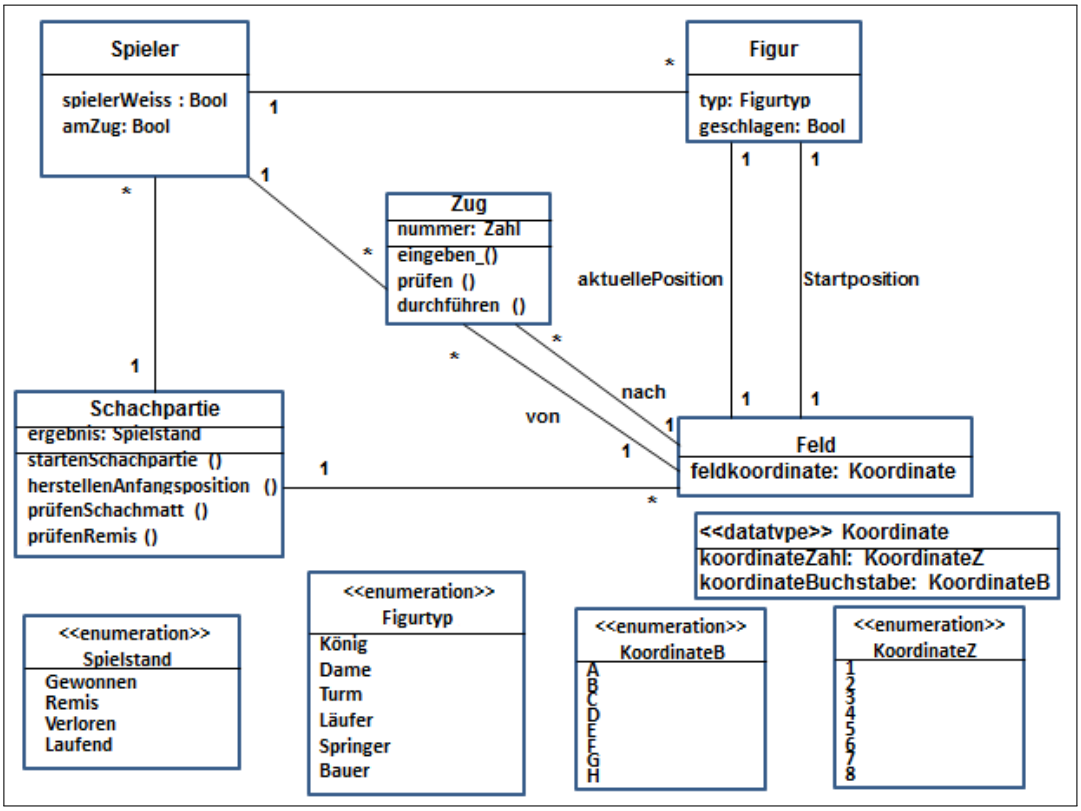


Abbildung A.3.: Musterlösung der Aufgabe „Schachpartie“

A.2.1.4. Fahrradverleih

Die Firma „Rent a Bike“ bietet Fahrräder zur Vermietung an. Dazu stehen Verleihstationen bereit, an denen Fahrräder ausgeliehen und zurückgegeben werden können. Jede Station besitzt eine Adresse. Ein Fahrrad besitzt eine eindeutige Nummer zur Identifizierung und einen Zustand (ausgeliehen, verfügbar, reserviert). Um ein Fahrrad ausleihen zu können, muss man sich mit Name, Adresse, Email und einem Passwort auf der Webseite des Anbieters registrieren. Registrierte Kunden können ein verfügbares Fahrrad an einer Station ausleihen. Ein Fahrrad muss an derselben Station zurückgegeben werden, wo es auch ausgeliehen wurde. Dabei wird der Rückgabezeitpunkt gespeichert. Die Kosten der Ausleihe richten sich nach deren Dauer und dem für alle Fahrräder gleichen Tagessatz. Über die Webseite des Betreibers sind folgende Funktionen verfügbar: Kunden bekommen eine detaillierte monatliche Rechnung erstellt mit Dauer, Kosten und den beteiligten Stationen jeder Ausleihe. Für einen Zeitpunkt in der Zukunft kann ein Fahrrad an einer gewünschten Station für eine bestimmte Dauer reserviert werden. Außerdem ist die Anzahl verfügbarer Fahrräder an jeder Station einsehbar.

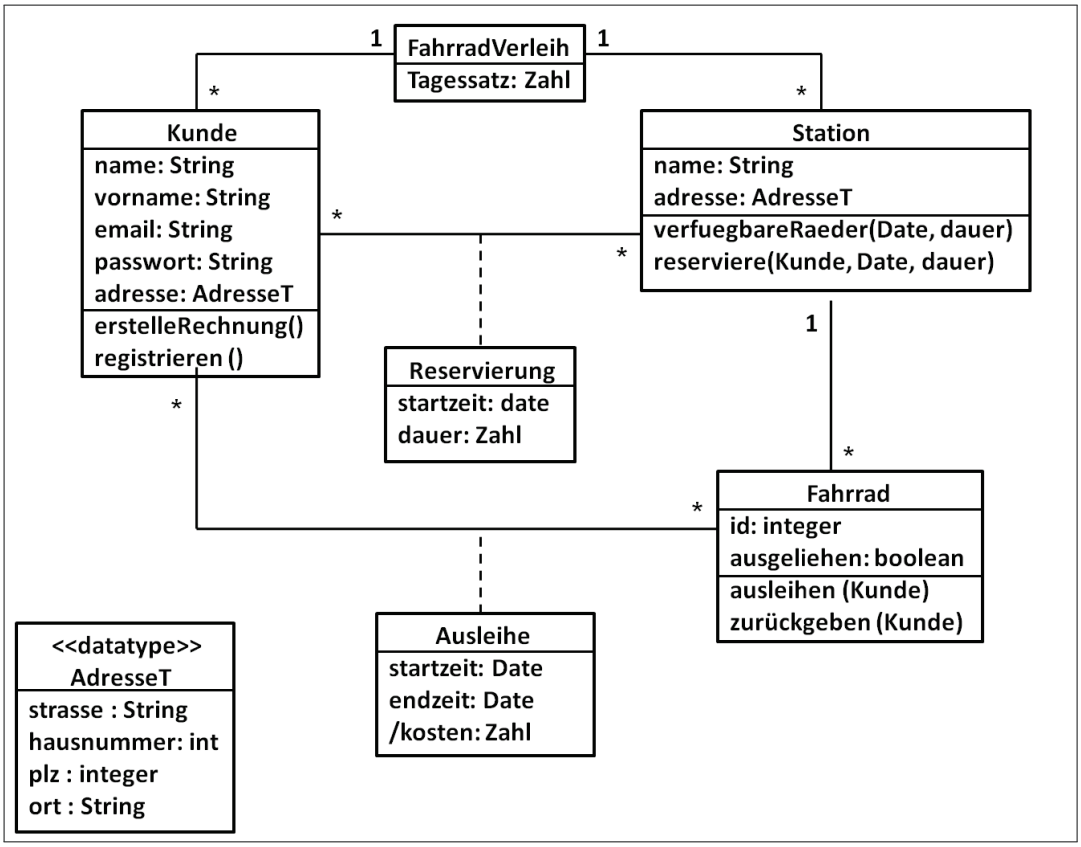


Abbildung A.4.: Musterlösung der Aufgabe „Fahrradverleih“

A.2.1.5. Landtagswahl

Bei einer Landtagswahl treten die zugelassenen Parteien und deren Kandidaten zur Wahl an, wobei es für jeden Stimmkreis einen Kandidaten für jede Partei gibt. Kandidaten haben einen Vornamen und einen Nachnamen. Jeder Stimmkreis hat eine Nummer und jede Partei besitzt einen Namen sowie eine Abkürzung. Bei der Auszählung der Wahl werden zwei Stimmzettel unterschieden: die Erststimme zur Wahl eines Kandidaten und die Zweitstimme zur Wahl einer Partei. Stimmzettel können ungültig sein, sie zählen dann trotzdem zur Wahlbeteiligung. Der Kandidat mit den meisten Erststimmen eines Stimmkreises gewinnt diesen Stimmkreis. Die Gesamtstimmenanzahl einer Partei ergibt sich aus der Summe der Zweitstimmen für die Partei und der Summe der Erststimmen aller Kandidaten der Partei. Parteien, die mehr als 5 % der Gesamtstimmen erhalten haben, ziehen in das Parlament ein. Das Parlament besitzt eine Grundanzahl von Sitzen und kennt die Anzahl der Wahlberechtigten. Durch Überhangmandate kann sich die Zahl der tatsächlich vergebenen Sitze im Parlament erhöhen. Aufgrund der Stimmanteile aller Parteien werden ihre Mandate und Überhangmandate berechnet.

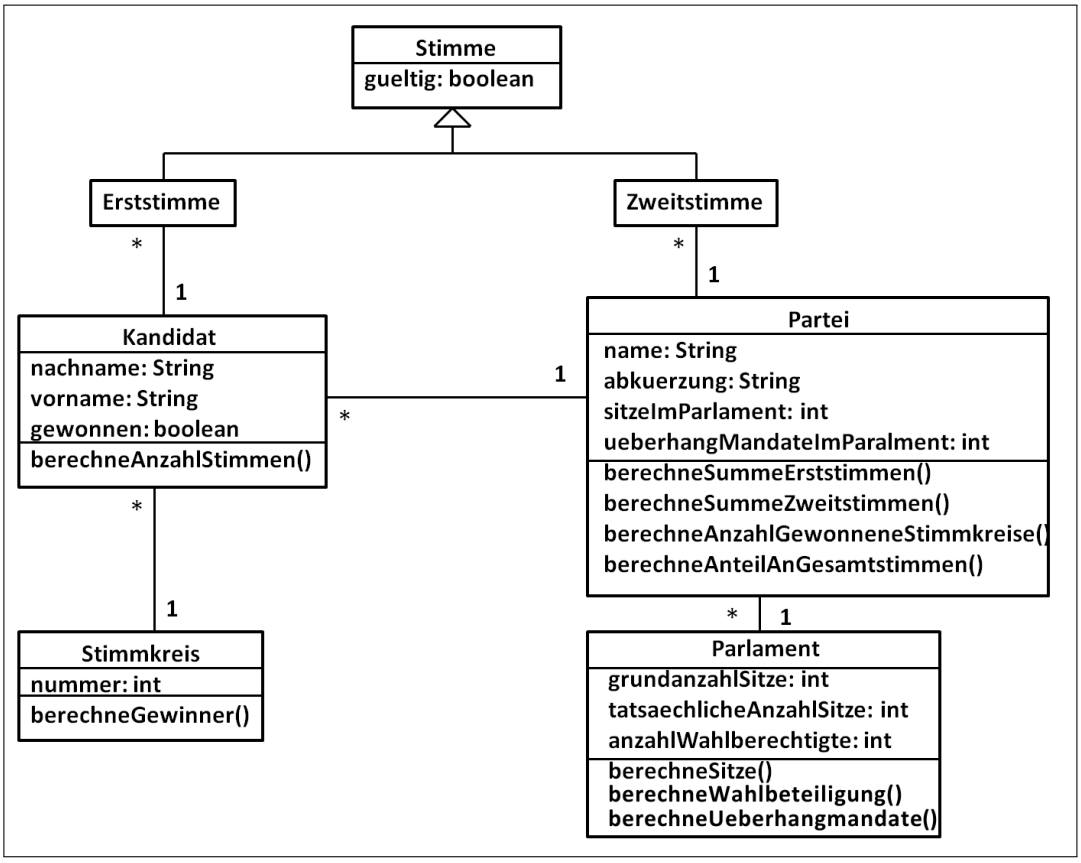


Abbildung A.5.: Musterlösung der Aufgabe „Landtagswahl“

A.2.1.6. Brettspiel Monopoly

Zu einem Monopoly-Spiel gehören ein Spielfeld und mehrere Spieler. Für jeden Spieler werden der Name und die Menge an Spielgeld, die er gerade besitzt vermerkt. Ein Spieler kann bei bestimmten Aktivitäten von der Bank oder Mitspielern Geld erhalten oder muss diesen Geld bezahlen. Ein Spielfeld besteht aus einer Reihe von Feldern. Diese Felder besitzen auf dem Spielfeld eine feste Reihenfolge und sind nummeriert. In unserer vereinfachten Variante gibt es drei verschiedene Arten von Feldern:

Das Losfeld (Startfeld), Ereigniskartenfelder und Straßenfelder. Zu jedem Straßenfeld gehört eine Straße, die von maximal einem Spieler besessen werden kann. Die Spieler würfeln nun reihum und ziehen Ihre Spielfigur auf dem Spielfeld um eine entsprechende Anzahl Felder nach vorne.Wenn er dabei über das Losfeld zieht, bekommt er einen bestimmten Betrag gutgeschrieben. Je nachdem, auf welchem Feld der Spieler landet, wird eine Aktion ausgeführt. Auf einem Ereigniskartenfeld zieht er je nach Typ des Feldes eine Karte von einem der mehreren Ereigniskartenstapel. Ein Ereigniskartenstapel besteht seinerseits wiederum aus einer Menge von Ereigniskarten, auf denen jeweils eine Anweisung steht, welche Sie nicht näher spezifizieren brauchen.

Landet der Spieler auf dem Feld einer Straße, die noch kein Spieler besitzt, so kann er diese (von der Bank) kaufen. Landet er auf dem Feld der Straße eines Mitspielers, so muss er Miete an diesen Mitspieler bezahlen, sofern diese nicht durch eine Hypothek belastet ist. Die Höhe der Miete berechnet sich aus einer Grundmiete und der Anzahl der Häuser auf dieser Straße. Hypotheken auf eigene Straßen kann der Spieler jederzeit während seines eigenen Zuges aufnehmen und zurückzahlen, um sich Geld von der Bank zu leihen. Der Kaufpreis und die Mietkosten sind für jede Straße einzeln festgelegt.

Zusätzlich gehören Straßen jeweils genau einer Farbgruppe an. Alle Häuser auf Straßen einer Farbgruppe haben den gleichen Preis. Häuser und Straßen verkaufen kann ein Spieler jederzeit während seines Zuges. Kann ein Spieler Mietkosten an einen anderen Spieler nicht bezahlen, so muss er ein oder mehrere seiner Straßen an die Mitspieler versteigern oder das Spiel aufgeben

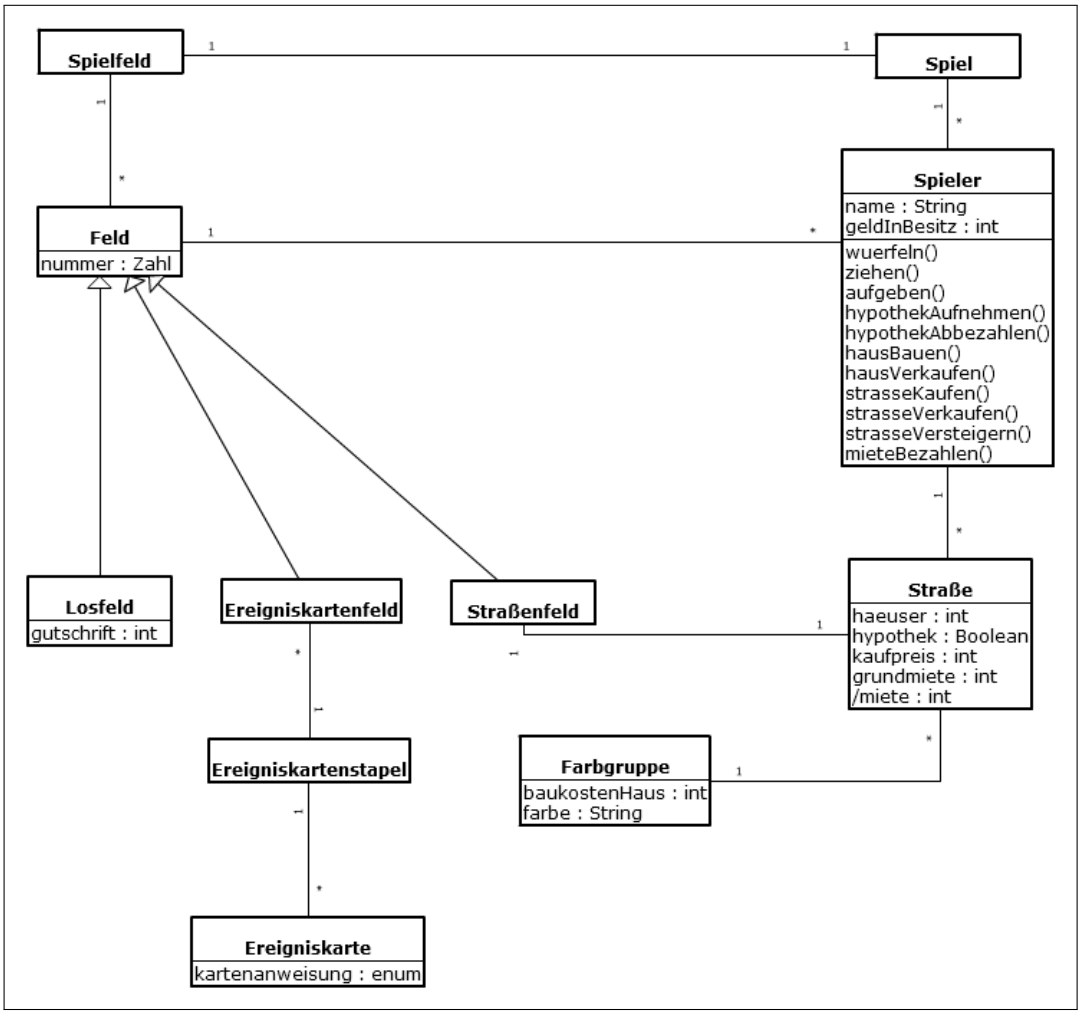


Abbildung A.6.: Musterlösung der Aufgabe „Brettspiel Monopoly“

.

A.2.1.7. Olympische Spiele (Sommersemester 2013)

In einem Softwaresystem sollen die folgenden Informationen zu Olympischen Spielen eingegeben, verarbeitet und angefragt werden können: An den Olympischen Spielen sind verschiedene Länder beteiligt, die durch ihren jeweiligen Namen identifiziert werden. Für jedes Land starten Athleten bei den Spielen, die in verschiedenen Wettkämpfen gegeneinander antreten. Ein Athlet kann dabei auch an mehreren Wettkämpfen teilnehmen. Für jeden Wettkampf werden der Austragungsort und das Wettkampfdatum festgehalten. Als Ergebnis eines Wettkampfes werden für jeden Athleten die Platzierung (Zahl) und die dadurch errungene Medaille (Gold, Silber, Bronze oder keine) gespeichert. Die Anzahl von Gold-, Silber und Bronze eines Landes ergibt sich aus der Summe der Medaillen der verschiedenen Sportler dieses Landes. Für jeden Sportler werden Nachname, Vorname, Geburtstag, Gewicht, Größe und sein Herkunftsland im System erfasst. Die Sportler werden von verschiedenen Betreuern versorgt. So können sich beliebig viele Trainer, Ärzte und Physiotherapeuten um einen oder mehrere Athleten kümmern. Für jeden Betreuer sollen Nachname, Vorname, Geburtstag und Herkunftsland gespeichert werden (die drei Typen von Betreuer sollen als Klassen modelliert werden).

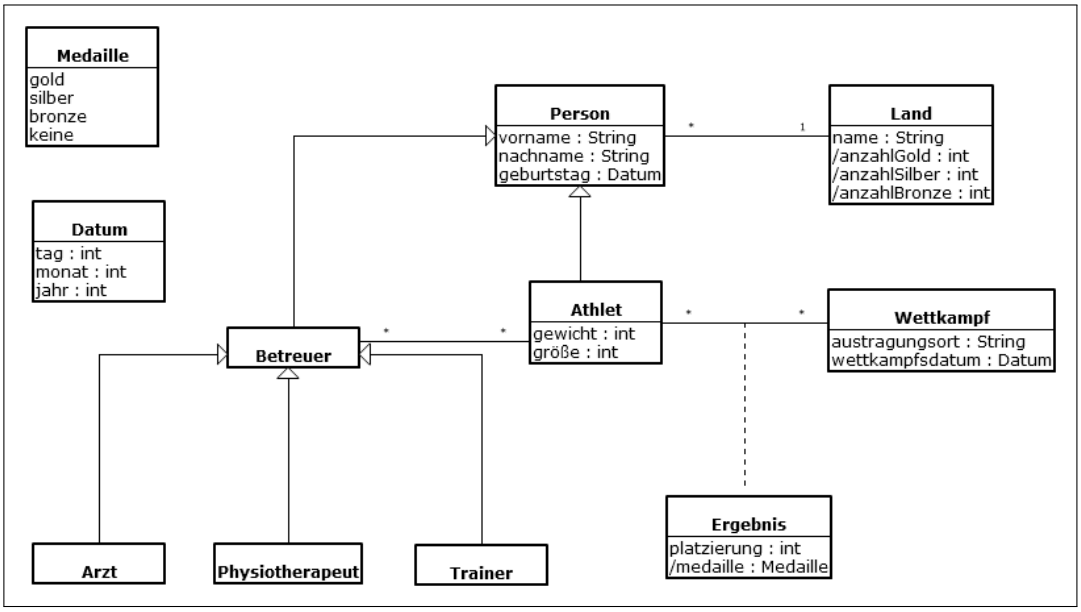


Abbildung A.7.: Musterlösung der Aufgabe „Olympische Spiele“ (2013)

A.2.1.8. Olympische Spiele (Sommersemester 2014)

An den Olympischen Spielen sind verschiedene Länder beteiligt, die durch ihren jeweiligen Namen identifiziert werden. Für jedes Land starten Athleten bei den Spielen, die

in verschiedenen Wettkämpfen gegeneinander antreten. Ein Athlet kann dabei auch an mehreren Wettkämpfen teilnehmen. Für jeden Wettkampf werden der Austragungsort und das Wettkampfdatum festgehalten. Als Ergebnis eines Wettkampfes werden für jeden Athleten die Platzierung und die dadurch errungene Medaille (Gold, Silber, Bronze oder keine) gespeichert. Die jeweiligen Medaillen eines Landes ergeben sich aus der Summe der Medaillen der verschiedenen Sportler. Für jeden Athleten werden Nachname, Vorname, Geburtstag, Gewicht, Größe und sein Herkunftsland im System erfasst. Die Athleten werden von verschiedenen Betreuern versorgt. Für jede Betreuungsperson sollen Nachname, Vorname, Geburtstag, Herkunftsland und Betreuungsart gespeichert werden, wobei es für die Betreuungsart nur 3 verschiedene Werte gibt, nämlich Arzt, Physiotherapeut und Trainer.

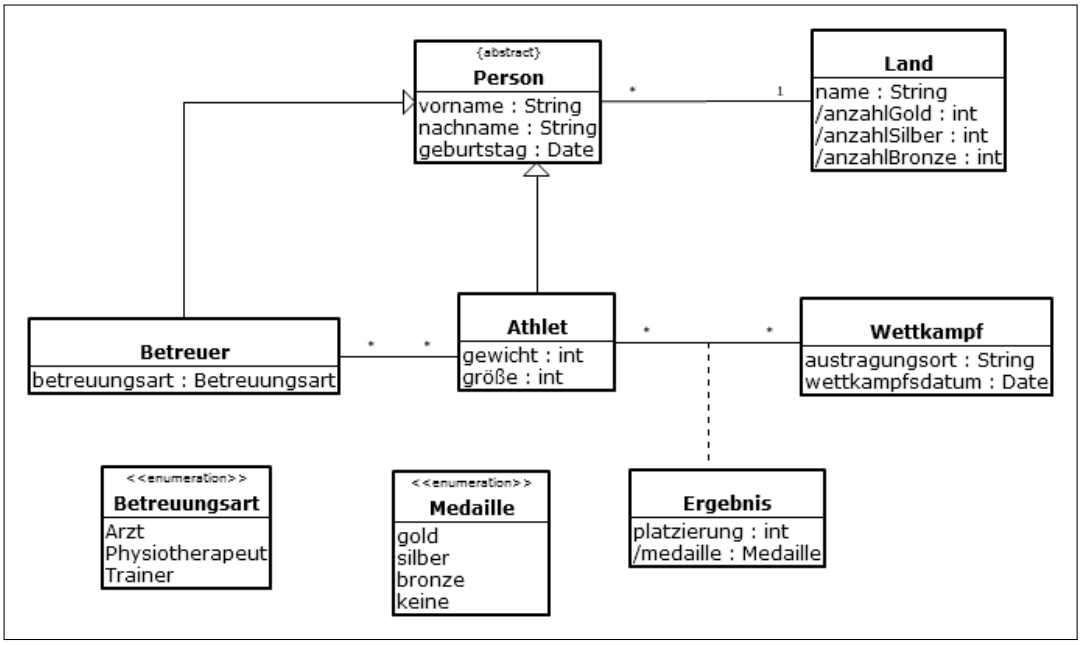


Abbildung A.8.: Musterlösung der Aufgabe „Olympische Spiele“ (2014)

A.2.2. UML Klassendiagramme: Evaluation der Zuordnungen

Liste aller 67 verschiedenen, korrekten (richtig positiven) Zuordnungen von Klassen bei der Aufgabe „Olympia“ im Sommersemester 2013:

Klasse Musterlösung	**Klasse Lernerlösung**
Arzt	Ärzte
Arzt	Artzt
Arzt	Aerzte
Athlet	Sportler(Athelete)
Athlet	Sportler
Athlet	Atlethen

Klasse Musterlösung	Klasse Lernerlösung
Athlet	Atleht
Athlet	Athleten/Sportler
Athlet	Athleten
Athlet	athlaten
Betreuer	Betruer
Betreuer	Betreuter
Betreuer	Betreuern
Betreuer	Betreuer abstract
Betreuer	Betreuen
Betreuer	Beträuer
Betreuer	Beteuer
Datum	Date
Ergebnis	Wettkampfteilnahme
Ergebnis	Wettkampfergebnis
Ergebnis	Wettkampf Teilnahme
Ergebnis	Platzierung
Ergebnis	Platz
Ergebnis	Ergenis
Ergebnis	Ergebniss
Ergebnis	Ergebnisliste
Ergebnis	Ergebnise
Ergebnis	Ergebnis<Athlet.Platzierung>
Ergebnis	BEWERTUNG
Ergebnis	«datatype» Ergebnis
Ergebnis	«Association Class» Ergebnis
Land	Länder
Medaille	Resultat
Medaille	Medallien
Medaille	Medallie
Medaille	Medalie
Medaille	Medaillie
Medaille	MedailleT
Medaille	Medaillentyp
Medaille	Medaillen
Medaille	Medailla
Medaille	Medaile
Medaille	Colour
Medaille	«enum» Medaillentyp
Person	Sportler
Person	Spieler
Person	Personen
Person	Landsmann
Physiotherapeut	Pysiotherapeut

Klasse Musterlösung	Klasse Lernerlösung
Physiotherapeut	Physiottherapeut
Physiotherapeut	Physiotherapheut
Physiotherapeut	Physiotherapeuthen
Physiotherapeut	Physiotherapeuth
Physiotherapeut	Physiotherapeuten
Physiotherapeut	Physiotherapen
Physiotherapeut	Physioterapheut
Physiotherapeut	Physioterapeut
Physiotherapeut	Phyiotherapeut
Physiotherapeut	Phisiotherapeuten
Physiotherapeut	Phisiotherapeut
Trainer	Trainier
Wettkampf	Wettkämpfen
Wettkampf	Wettkämpfe
Wettkampf	Wettkaempfen
Wettkampf	Weltkampf
Wettkampf	Spiele
Wettkampf	Spiel

Liste aller 10 verschiedenen, unkorrekten (falsch positiven) Zuordnungen von Klassen bei der Aufgabe „Olympia“ im Sommersemester 2013:

Klasse Musterlösung	Klasse Lernerlösung
Ergebnis	Sportler
Ergebnis	Medaillenspiegel
Ergebnis	Gold
Medaille	Summer der Medaillen eines Landes
Medaille	Summe der Medaillen eines Landes
Medaille	Summe der Medaillen
Medaille	Medaillenspiegel
Medaille	Liste
Person	Austragungsort
Person	Ärtzte

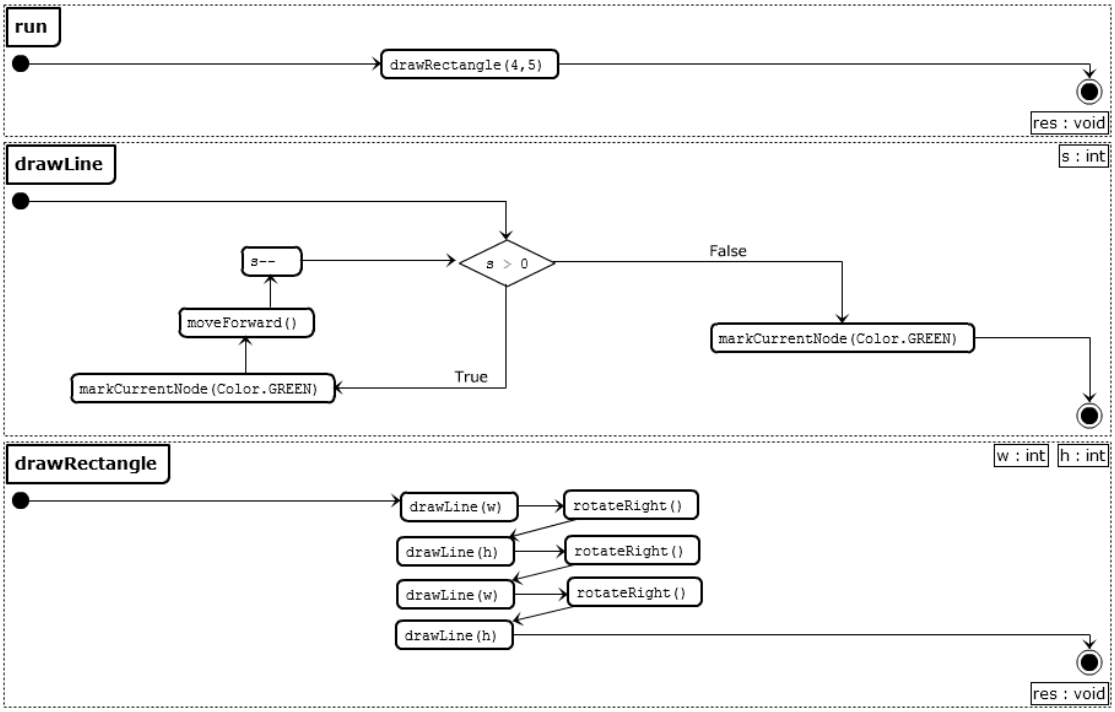
Liste aller 14 verschiedenen, fälschlicherweise nicht vorgenommenen (falsch negativen) Zuordnungen von Klassen bei der Aufgabe „Olympia“ im Sommersemester 2013:

Klasse Musterlösung	Klasse Lernerlösung
Arzt	Ärtzte
Arzt	Ärtze
Athlet	Sportler
Datum	Wettkampdatum

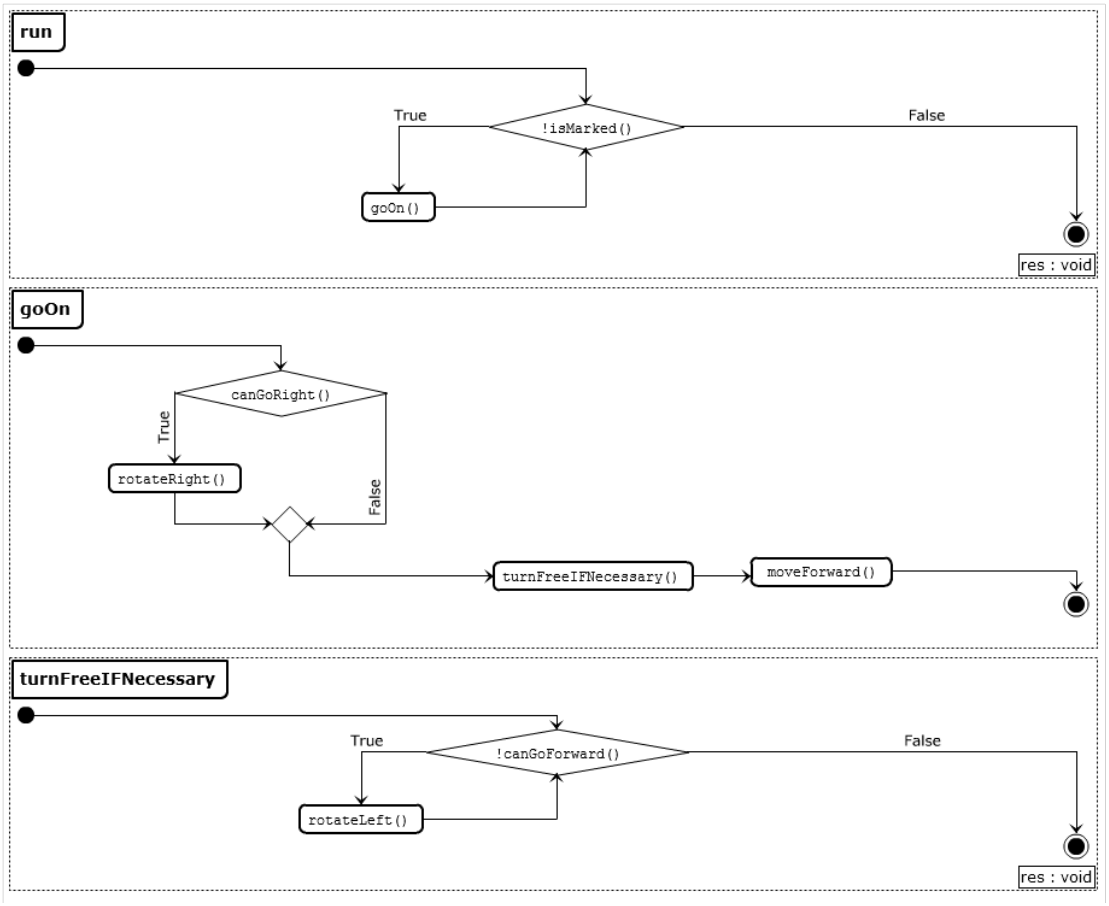
Klasse Musterlösung	Klasse Lernerlösung
Ergebnis	Wettkampf-Platzierung
Medaille	Wettkampfergebnis
Physiotherapeut	Therapeuten
Physiotherapeut	Pyhsiothreapeut
Physiotherapeut	Psychotherapeuten
Physiotherapeut	Psychoterapeuten
Physiotherapeut	Psychiotherapeuten
Physiotherapeut	Physiotherapien
Physiotherapeut	Physio
Wettkampf	Wettbewerb

A.2.3. UML Aktivitätsdiagramme: Musterlösungen

A.2.3.1. Aufgabe „Rechteck“



A.2.3.2. Aufgabe „Labyrinth“ bzw. „LabyrinthRandom“



A.2.4. Übungsblätter

A.2.4.1. Übungsblatt 03 im Sommersemester 2013

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Softwaretechnik
SoSe 2013

Übungsblatt 03

(50 Punkte)
Prof. F. Puppe, Prof. J. Wolff v. Gudenberg, M. Witek, G. Dietrich, M. Ertl, D. Wieth

Ausgabe: 07.05.2013
Abgabe: 15.05.2013, 12:00 (elektronisch)

Aufgabe 4: Erstellung von Klassendiagrammen (20 Punkte)

Lösen Sie folgende zwei Aufgaben mittels eines frei gezeichneten Klassendiagramms mit CaseTrain, das Sie über den WueCampus-Kurs aufrufen können.

4a) Zeichen Sie ein Klassendiagramm für die Aufgabe "Monopoly" (10 Punkte). Dabei sind die zu modellierenden Klassen vorgegeben und Sie sollen dazu Assoziationen, Attribute (ggf. mit DataTyes ergänzen) und Methoden hinzufügen. Starten Sie die Aufgabe durch Klick auf den Link "Übung 3: UML-Klassendiagramm Monopoly" im WueCampus-Kurs.

Gegeben seien folgende Informationen zu einem Monopoly-Spiel. Um die Modellierung zu erleichtern wurden viele stark vereinfachende Annahmen getroffen. Beschränken Sie in Ihrem eigenen Interesse Ihre Modellierung auf die folgende Beschreibung:
Zu einem Monopoly-Spiel gehören ein Spielfeld und mehrere Spieler. Für jeden Spieler werden der Name und die Menge an Spielgeld, die er gerade besitzt vermerkt. Ein Spieler kann bei bestimmten Aktivitäten von der Bank oder von Mitspielern Geld erhalten oder er muss diesen Geld bezahlen.
Ein Spielfeld besteht aus einer Reihe von Feldern. Diese Felder haben auf dem Spielfeld eine feste Reihenfolge und sind nummeriert. In unserer vereinfachten Variante gibt es drei verschiedene Arten von Feldern: Das „Losfeld“ (Startfeld), Ereigniskartenfelder und Straßenfelder. Zu jedem Straßenfeld gehört eine Straße, die von maximal einem Spieler besessen werden kann.
Die Spieler würfeln nun reihum und ziehen Ihre Spielfigur auf dem Spielfeld um eine entsprechende Anzahl Felder nach vorne. Wenn er dabei über das Losfeld zieht, bekommt er einen bestimmten Betrag gutgeschrieben. Für das Feld, auf dem der Spieler landet, führt er die zugehörige Aktion aus: Direkt auf dem Losfeld nimmt er das doppelte Geld ein. Auf einem Ereigniskartenfeld zieht er je nach Typ des Feldes eine Karte von einem der mehreren Ereigniskartenstapel. Ein Ereigniskartenstapel besteht seinerseits wiederum aus einer Menge von Ereigniskarten, auf denen jeweils eine Anweisung steht, welche Sie nicht näher spezifizieren brauchen. Landet der Spieler auf dem Feld einer Straße, die noch kein Spieler besitzt, so kann er diese (von der Bank) kaufen. Landet er auf dem Feld der Straße eines Mitspielers, so muss er „Miete“ an diesen Mitspieler bezahlen, sofern diese nicht durch eine Hypothek belastet ist. Die Höhe der Miete berechnet sich aus einer Grundmiete und der Anzahl der Häuser auf dieser Straße (wobei der Einfachheit halber angenommen wird, dass sich die Miete aus 10 * Grundmiete * Anzahl der Häuser ergibt). Hypotheken auf eigene Straßen kann der Spieler jederzeit während seines eigenen Zuges aufnehmen und zurückzahlen, um sich Geld von der Bank zu leihen. Der Kaufpreis und die Mietkosten sind für jede Straße einzeln festgelegt.
Zusätzlich gehören Straßen jeweils genau einer Farbgruppe an. Besitzt ein Spieler alle Straßen einer Farbgruppe, so kann er während seines Zuges auf Straßen dieser Farbgruppe bis zu fünf Häuser bauen. Alle Häuser auf Straßen einer Farbgruppe haben den gleichen Preis. Häuser und Straßen kann ein Spieler jederzeit während seines Zuges verkaufen. Kann ein Spieler die Mietkosten an einen anderen Spieler nicht bezahlen, so muss er ein oder mehrere seiner Straßen an die Mitspieler versteigern oder das Spiel aufgeben.

4b) Zeichnen Sie frei ein Klassendiagramm für die Aufgabe "Olympia" (10 Punkte). Halten Sie sich dabei streng an die im Text vorkommenden Namen für Klassen und Attribute und benutzen Sie abgeleitete Attribute soweit möglich (statt Methoden). Starten Sie die Aufgabe durch Klick auf den Link "Übung 3: UML-Klassendiagramm Olympia" im WueCampus-Kurs.

In einem Softwaresystem sollen die folgenden Informationen zu Olympischen Spielen eingegeben, verarbeitet und angefragt werden können:

An den Olympischen Spielen sind verschiedene Länder beteiligt, die durch ihren jeweiligen Namen identifiziert werden. Für jedes Land starten Athleten bei den Spielen, die in verschiedenen Wettkämpfen gegeneinander antreten. Ein Athlet kann dabei auch an mehreren Wettkämpfen teilnehmen. Für jeden Wettkampf werden der Austragungsort und das Wettkampfdatum festgehalten. Als Ergebnis eines Wettkampfes werden für jeden Athleten die Platzierung (Zahl) und die dadurch errungene Medaille (Gold, Silber, Bronze oder keine) gespeichert. Die Anzahl von Gold-, Silber- und Bronzemedaillen eines Landes ergibt sich aus der Summe der Medaillen der verschiedenen Sportler dieses Landes. Für jeden Sportler werden Nachname, Vorname, Geburtstag, Gewicht, Größe und sein Herkunftsland im System erfasst. Die Sportler werden von verschiedenen Betreuern versorgt. So können sich beliebig viele Trainer, Ärzte und Physiotherapeuten um einen oder mehrere Athleten kümmern. Für jeden Betreuer sollen Nachname, Vorname, Geburtstag und Herkunftsland gespeichert werden (die drei Typen von Betreuer sollen als Klassen modelliert werden).

Aufgabe 5: Erstellung von Aktivitätsdiagrammen (30 Punkte)

Lösen Sie folgende drei Aufgaben mittels eines Aktivitätsdiagramms mit CaseTrain-WARP, das Sie über den WueCampus-Kurs aufrufen können. Bevor Sie die Aufgaben lösen, empfehlen wir Ihnen das Video zur Bedienung anzuschauen und mittels Klick auf den Link "CaseTrain-WARP Leer" im WueCampus-Kurs selbst einzugeben.

5a) Definieren Sie in einem Aktivitätsdiagramm eine generische Methode "zeichneRechteck(int laenge, int breite)", die ein Rechteck der Länge laenge und der Breite breite zeichnet. Starten Sie diese Methode mit der Default-Aktivität "run()", die zeichneRechteck(4, 5) aufruft. Starten Sie die Aufgabe durch Klick auf den Link "CaseTrain-WARP Rechteck" im WueCampus-Kurs.

5b) Definieren Sie in einem Aktivitätsdiagramm eine generische Methode "staubsaugen()", die das bestehende Feld komplett saugen, d. h. alle Einzelfelder einfärben soll. Die Länge und Breite des Feldes ist nicht bekannt. In der Ausgangssituation steht der Roboter in der linken oberen Ecke des Feldes und schaut nach rechts. Starten Sie diese Methode mit der Default-Aktivität "run()", die staubsaugen() aufruft. Starten Sie die Aufgabe durch Klick auf den Link "CaseTrain-WARP Hoover" im WueCampus-Kurs.

5c) Definieren Sie in einem Aktivitätsdiagramm eine generische Methode "findeSchatz()", die in einem unbekannten Labyrinth einen Schatz findet. Der Schatz liegt auf einem farbigen Einzelfeld, das Sie mit der Methode isMarked() entdecken, d.h. isMarked() liefert auf dem Feld, auf dem sich der Schatz befindet, "true" zurück und sonst "false". In der Ausgangssituation steht der Roboter in der linken oberen Ecke des Feldes und schaut nach rechts. Starten Sie diese Methode mit der Default-Aktivität "run()", die findeSchatz() aufruft. Starten Sie die Aufgabe durch Klick auf den Link "CaseTrain-WARP Labyrinth" im WueCampus-Kurs.

Insgesamt stehen Ihnen bei den Aufgaben 5a-5c folgende Systemmethoden zur Verfügung:

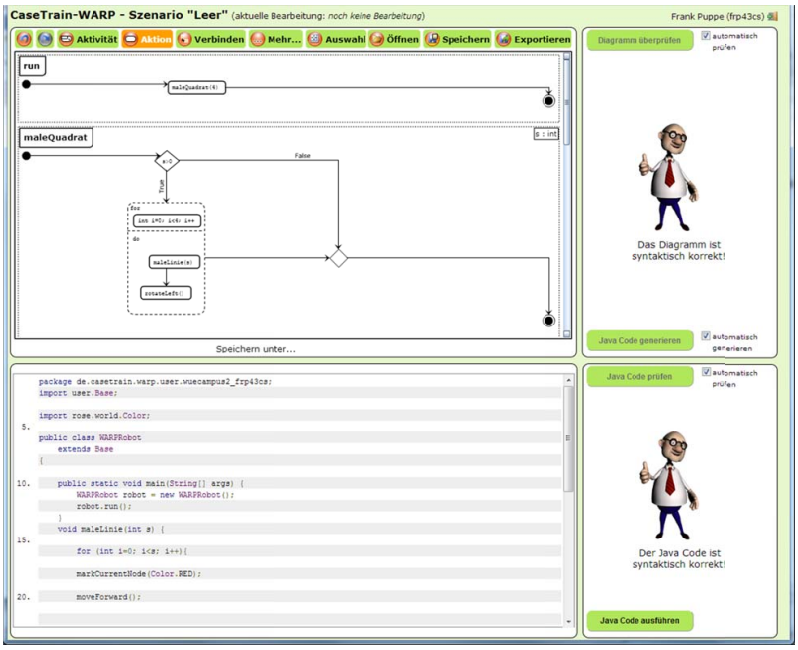
```
moveForward()              -- gehe ein Feld vor
rotateRight()              -- drehe dich nach links
rotateLeft()               -- drehe dich nach rechts
moveRandom()               -- gehe ein Feld vor mit 50% Wahrscheinlichkeit, ansonsten drehe dich mit
                           jeweils 25% Wahrscheinlichkeit in eine  Richtung und  gehe dann ein Feld vor

canGoForward(): true/false -- kann ich nach vorne gehen?
canGoLeft(): true/false    -- kann ich nach links gehen?
```

canGoRight(): true/false -- kann ich nach rechts gehen?
isMarked(): true/false -- ist der Knoten, auf dem ich gerade stehe, mit einer Farbe markiert?

markCurrentNode(rose.world.Color c) -- markiere den Knoten unter dir mit Farbe c
-- zum Beispiel
-- markCurrentNode(Color.GREEN)

Überprüfen Sie die Korrektheit der Aufgaben, indem Sie zunächst prüfen, ob der Aktivitätsdiagramm-Code korrekt ist (dann produziert er korrekten Java-Code) und dann den Java-Code ausführen (s.u. für die Aufgabe maleQuadrat(s).



A.2.4.2. Übungsblatt 04 im Sommersemester 2014

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Softwaretechnik

SoSe 2014

Übungsblatt 04

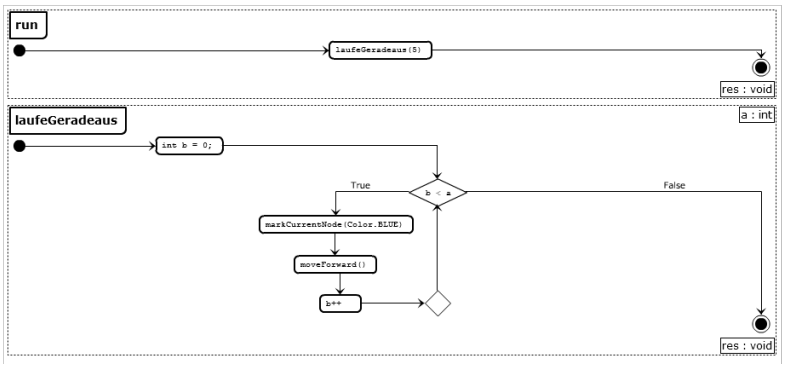
(40 Punkte)

Prof. F. Puppe, Prof. S. Kounev, J. Walter, G. Dietrich, M. Ertl,

Ausgabe: 08.05.2014
Abgabe: 16.05.2014, 6:00 Uhr (elektronische Abgabe; einzeln, d.h. nicht in Gruppen)

Aufgabe 5: Aktivitätsdiagramme (40 Punkte)

Lösen Sie folgende Aufgaben jeweils mittels der Eingabe eines Aktivitätsdiagramms in CaseTrain-WARP, welches das Verhalten eines Roboters definiert. Das folgende Beispiel zeigt das aus zwei Aktivitäten bestehende Aktivitätsdiagramm eines Roboters, der 5 Felder geradeaus läuft und diese dabei blau markiert.



Bevor Sie die Aufgaben lösen, empfehlen wir Ihnen, die Kurzanleitung zu CaseTrain-WARP zu lesen, die Screencasts zur Bedienung anzuschauen (Links im WueCampus-Kurs), das obige Diagramm selbst einzugeben und im Szenario „Leer“ zu testen. Starten Sie dazu CaseTrain-WARP über den Link „CaseTrain-WARP: Szenario Leer“ im WueCampus-Kurs.

Zu jeder Aufgabe gehört ein sogenanntes Szenario. Die Erstellung der Aktivitätsdiagramme für das jeweilige Szenario starten Sie durch Klick auf den entsprechenden Link "CaseTrain-WARP: Aufgabe *Aufgabenname*" im WueCampus-Kurs. Testen Sie Ihre Roboter ausgiebig in der „WARP Arena“, die Sie u.a. über den Link „CaseTrain-WARP: Arena“ erreichen.
Die Einreichung eines Aktivitätsdiagramms erfolgt durch Eingabe des bei dessen Erstellung automatisch generierten Roboter-Keys auf der Einreichungs-Seite, welche Sie u.a. über den Link „CaseTrain-WARP: Einreichung“ im WueCampus-Kurs erreichen. Achten Sie hier darauf, dass Sie das korrekte Szenario wählen! Sie können je Aufgabe mehrere Einreichungen vornehmen, wobei ausschließlich die neueste Einreichung bewertet wird. Wenn Sie für eine Szenario mehr als einen Roboter Einreichen, wird Ihnen pro zusätzlicher Einreichung 1 Punkt abgezogen. Testen Sie Ihre Roboter zuvor also ausgiebig in der WARP-Arena!

5a) Rechteck (5 Punkte)
Definieren Sie in einem Aktivitätsdiagramm eine generische Methode `zeichneRechteck(int laenge, int breite)`, die ein Rechteck der Länge `laenge` und der Breite `breite` zeichnet. Starten Sie diese Methode mit der Default-Aktivität `run()`, die `zeichneRechteck(4, 5)` aufruft.

5b) Hoover (5 Punkte)

Definieren Sie in einem Aktivitätsdiagramm eine generische Methode `staubsaugen()`, die das bestehende Feld komplett saugt, d. h. alle Einzelfelder einfärben soll. Die Länge und Breite des Feldes ist nicht bekannt. In der Ausgangssituation steht der Roboter in der linken oberen Ecke des Feldes und schaut nach rechts. Starten Sie diese Methode mit der Default-Aktivität `run()`, die `staubsaugen()` aufruft.

5c) HooverRandom (10 Punkte)
Gehen sie analog zu Aufgabe b) vor. In diesem Szenario befinden sich 3 zufällig auf dem Feld verteilte Hindernisse, die der Roboter nicht befahren kann (und somit auch nicht markiert werden müssen).

5d) LabyrinthRandom (10 Punkte)
Definieren Sie in einem Aktivitätsdiagramm eine generische Methode `findeSchatz()`, die in einem unbekannten Labyrinth einen Schatz findet. Starten Sie diese Methode mit der Default-Aktivität `run()`, die `findeSchatz()` aufruft. Der Schatz liegt auf einem farbigen Einzelfeld, das Sie mit der Methode `isMarked()` entdecken, d.h. `isMarked()` liefert auf dem Feld, auf dem sich der Schatz befindet, `true` zurück und sonst `false`. In der Ausgangssituation steht der Roboter auf einem zufälligen Feld.

5e) Kaninchenjagd (10 Punkte)
Definieren Sie in einem Aktivitätsdiagramm einen jagenden Roboter mit einer Default-Aktivität `run()`, der das auf dem Feld hoppelnde und während des Fressens unbewegte Kaninchen (blau) schneller finden und erlegen kann als der WARP-Standard-Jäger. Auf dem Feld befinden sich mehrere Bäume als Hindernisse, die weder von den Jägern, noch vom Kaninchen betreten werden können. Das Kaninchen kann nur dann mittels der Methode `shoot()` erlegt werden, wenn es direkt vor einem Jäger steht. Um das Kaninchen zu finden, stehen Ihnen weitere Methoden zur Verfügung (s.u.).
Alternativ können Sie diese Aufgabe auch durch direkte Eingabe vom Java-Quellcode lösen. Starten Sie die Eingabe durch einen Klick auf den Link "CaseTrain-WARP: Szenario Kaninchenjagd (Direkteingabe)“ im WueCampus Kurs.

Insgesamt stehen Ihnen bei den Aufgaben 5a-5e folgende vordefinierte Methoden für Ihren Roboter zur Verfügung:

`void moveForward()`	gehe ein Feld vor
`void rotateRight()`	drehe dich nach rechts
`void rotateLeft()`	drehe dich nach links
`void moveRandom()`	gehe ein Feld vor mit 50% Wahrscheinlichkeit, ansonsten drehe dich mit jeweils 25% Wahrscheinlichkeit in eine Richtung und gehe dann ein Feld vor
`void markCurrentNode(Color c)`	markiere das aktuelle Feld mit Farbe c Beispiel: `markCurrentNode(Color.GREEN)`
`boolean canGoForward()`	kann ich nach vorne gehen?
`boolean canGoLeft()`	kann ich nach links gehen?
`boolean canGoRight()`	kann ich nach rechts gehen?
`boolean isMarked()`	ist das Feld, auf dem ich gerade stehe, mit einer Farbe markiert?

für Aufgabe 5e) Kaninchenjagd:

`boolean thereIsARabbit()`	lebt das Kaninchen noch?
`int getForwardDistanceToRabbit()`	gebe zurück, wie viele Felder das Kaninchen vor mir steht
`int getBackwardDistanceToRabbit()`	gebe zurück, wie viele Felder das Kaninchen hinter mir steht
`int getLeftDistanceToRabbit()`	gebe zurück, wie viele Felder das Kaninchen links von mir steht
`int getRightDistanceToRabbit()`	gebe zurück, wie viele Felder das Kaninchen rechts von mir steht
`void shoot()`	erlegt das Kaninchen, wenn es direkt vor mir steht

Freiwillige Übungsaufgabe:

5f) KaninchenjagdMulti
In diesem Szenario können Sie Ihre in Aufgabe 5e) erstellten Roboter gegeneinander antreten lassen und herausfinden, wer der bessere Jäger ist.
Starten Sie dazu die CaseTrain-WARP Arena durch einen Klick auf den Link „CaseTrain-WARP: Arena KaninchenjagdMulti“ im WueCampus-Kurs und geben Sie Keys von selbst erstellten Robotern oder Keys von Robotern Ihrer Kommilitonen ein.

A.2.5. Fragebogen zu elektronischen Werkzeugen

A.2.5.1. Präsentation des Fragebogens im Sommersemester 2013

Bitte nehem Sie sich noch etwas Zeit und bewerten Sie die in den Übungen zur Vorlesung Softwaretechnik eingesetzten E-Learning-Werkzeuge insgesamt und nach verschiedenen Einzelkriterien mit Schulnoten!

	Note
1. Wie bewerten Sie den Einsatz von Tools im Vergleich zu konventionellem Übungsbetrieb prinzipiell?	
2. Wie bewerten Sie den Einsatz von Tools im Vergleich zu konventionellem Übungsbetrieb konkret für diese Vorlesung?	
3. Halten Sie es für sinnvoll, Tools im Übungsbetrieb in Zukunft weiter einzusetzen (1= ja / 0 = nein)?	

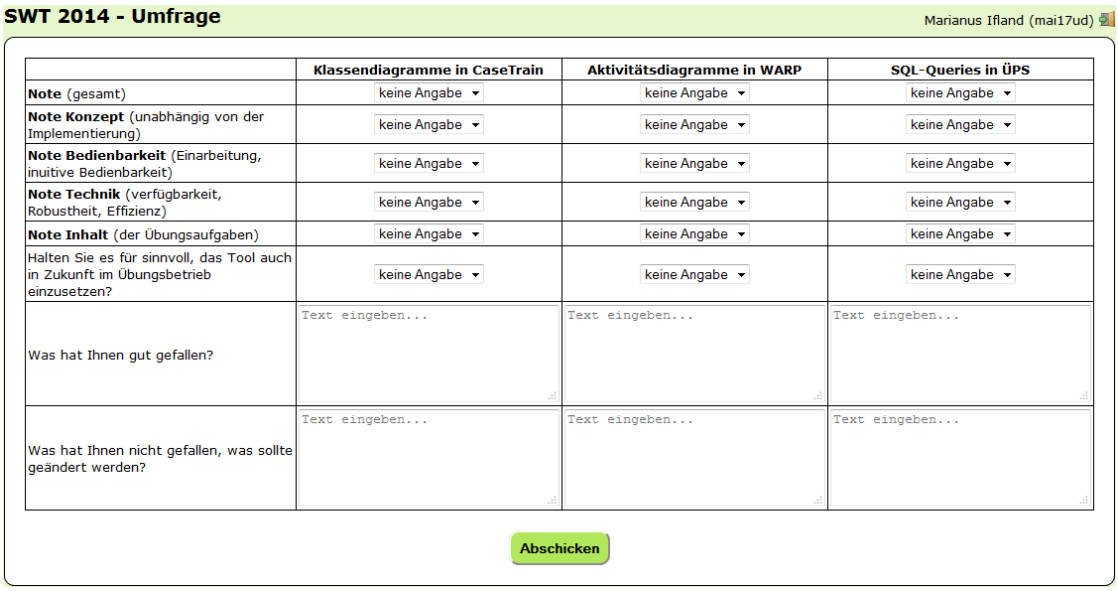
In dem Übungsbetrieb wurden unterschiedliche E-Learning-Tools eingesetzt. Bitte bewerten Sie diese einzeln nach verschiedenen Kriterien.

Bitte tragen Sie in jedes Kästchen eine Schulnote ein!	Allgemeine MC-Fragen in CaseTrain (Übung 1)	Pflichten-heft-Aufgaben (Übung 2)	MC-Fragen zu Klassen-diagrammen in CaseTrain (Übung 2)	Freies Zeichnen von Klassen-diagrammen (Übung 3)	Freies Zeichnen von Aktivitäts-diagrammen (Übung 3)	Eingabe von SQL-State-ments (Übung 4)
4. Konzept des Tools, unabhängig von der Implementierung						
5. Umsetzung des Tools, Gesamtnote						
6. Aspekt Bedienbarkeit (Einarbeitung, intuitive Bedienbarkeit)						
7. Aspekt Technik (Verfügbarkeit, Robustheit, Effizienz)						
8. Aspekt Inhalt der Übungsaufgaben						
9. Aspekt Fairniss der Bewertung der Übungsaufgaben						
10. Aspekt hilfreiche Erklärungen als Text oder nützliches Feedback z.B. bei Aktivitätsdiagramm/SQL-Aufgaben zur Lösung der Übungsaufgaben.						
Bitte kreuzen Sie an, wenn Sie die jeweilige Aufgabe per Email/Papier abgegeben haben.						

Ihr Kommentar (optional):

Abbildung A.9.: Papier-Fragebogen im Sommersemester 2013

A.2.5.2. Präsentation des Fragebogens im Sommersemester 2014



SWT 2014 - Umfrage Marianus Ifland (mai17ud)

	Klassendiagramme in CaseTrain	Aktivitätsdiagramme in WARP	SQL-Queries in ÜPS
Note (gesamt)	keine Angabe	keine Angabe	keine Angabe
Note Konzept (unabhängig von der Implementierung)	keine Angabe	keine Angabe	keine Angabe
Note Bedienbarkeit (Einarbeitung, inuitive Bedienbarkeit)	keine Angabe	keine Angabe	keine Angabe
Note Technik (verfügbarkeit, Robustheit, Effizienz)	keine Angabe	keine Angabe	keine Angabe
Note Inhalt (der Übungsaufgaben)	keine Angabe	keine Angabe	keine Angabe
Halten Sie es für sinnvoll, das Tool auch in Zukunft im Übungsbetrieb einzusetzen?	keine Angabe	keine Angabe	keine Angabe
Was hat Ihnen gut gefallen?	Text eingeben...	Text eingeben...	Text eingeben...
Was hat Ihnen nicht gefallen, was sollte geändert werden?	Text eingeben...	Text eingeben...	Text eingeben...

Abschicken

Abbildung A.10.: Online-Fragebogen im Sommersemester 2014

A.2.5.3. Umfrage-Ergebnisse für WARP im Sommersemester 2014

Hinweis: Die folgenden Freitext-Anwtorten wurden unverändert übernommen, also inklusive Fehlern in Rechtschhreibung und Interpunktion.

Antworten auf die Frage **„Was hat Ihnen gut gefallen?“**:

- „Praktische“ Anwendung zeigt schnell sichtbare Ergebnisse, macht Spaß.
- Meiner Meinung nach war es sehr hübsch aufgemacht und hat mir persönlich gefallen
- interaktiv, man sieht, was man macht
- -Leicht verständlich auch ohne Programmiervorkenntnisse in Java (Entmutigt nicht sofort, wenn man etwas nicht gleich kann)
  -Gute Einführung für einfache Programmieraufgaben
  -Keine Langweiligen Aufgaben
  -Durch die Animation lässt sich sein Diagramm leicht auf Richtigkeit überprüfen und man kann Schritt für Schritt sein Diagramm erweitern um die Aufgabe zu lösen
- Die Idee hinter der „interaktiven“ Programmierung finde ich sehr gut.
  Mit Beseitigung der Bugs/Bedienprobleme finde ich es aber eine sehr sinnvolle Übungsmethode, welche auch Spaß macht, da das Ergebnis direkt zu erkennen ist.

- verständliche Fehlermeldung
  direcktes grafisches Feedback
- Gut fand ich, dass man aus den Aktivitätsdiagrammen direkt Code erzeugen lassen konnte und somit auch gesehen hat, was das Programm schließlich tut. Jedoch war der Code mitunter auch falsch, etwa bei if-then-else-Statements, die man in dem strukturierten if-Block erzeugt hat.
- Ebenfalls ist die Anwendung sehr gut.
- Die allgemeine Verfügbarkeit finde ich sehr von Vorteil, da sie einem beim Verständnis von Aktivitätsdiagrammen hilft.
- Das der Lehrstuhl versucht neue Wege beim lernen zu gehen
- Man bekommt ein gutes Gefühl dafür, in wie Fern Aktivitätsdiagramme mit Code zusammenhängen
- Es war schön, in der Arena zu sehen, was man zuvor „erschaffen“ .
- Ist das Programm und die Bedienung erst eimal verstanden macht es durch aus spaß sich algorithmen zu überlegen und zu optimieren
- die steigende Komplexität der Aufgaben erleichtert das Verständnis und somit auch die Bearbeitung.
- Mir gefällt die Lösungsanimation des Programms.
- Sofortiges Feedback, man kann unendlich lange probieren und ggf. verbessern.
- Die Aufgaben waren motivierend und haben Spaß gemacht. Schade dass die Umsetzung noch starke mängel aufwies.
- Sehr schöne Idee und interessante und spaßige Aufgaben
- Grundidee, bzw. Inhalt der Aufgaben gut.
- Das Prinzip
- Man konnte seine Implementierung direkt überprüfen und so Optimierung anstreben
- Idee ist ganz cool.
- Die direkte Übersetzung in Code und deren Auswirkung auf ein „virtuell“ Lebendes Objekt um Wechselwirkung mit der Realität darzustellen
- man kann schnell und ohne sich um die Syntax kümnmern zu müssen ein funktionsfähiges Programm erzeugen.
- Aufgaben
- Das Konzept ist zur Übung zum Erstellen von UML - Diagrammen sehr sinnvoll!
- Keine Implementierung von Java-Code. Konkrete Aufgabenstellung.
- Die Idee

- ..dass es nach Abänderung auch auf dem Laptop nutzbar war
  ..die Aufgabe an sich finde ich sehr gut, weil man praktisch sehen kann, was man mit dem Programmieren erreicht
- Sehr motivierend spielerisch die Aufgaben zu erledigen und auch ein sichtbares Ergebnis zu produzieren. Macht die sonst eher trockene Materie interessant und spaßig. Auch die Ideen bei der Aufgabenstellung, z.B. Hasenjagd sehr kreativ.
- Die Idee ist hervorragend, kann den Stoff viel anschaulicher machen
- nen versuch war es wert. bei richtiger implementation könnte es ganz spasig sein
- - Gute Idee / Inhalte
  - Animationen sehr anschaulich
- - intensive Beschäftigung mit der Materie - direkte Ausgabe vom Code, man konnte sehen, was man 'geschaffen' hat
- - Einfache Umsetzung von Programmierideen ohne benötigte Programmierkenntnisse
- Gute Aufgaben, einfache Bedienung und in meinem Fall keine Probleme mit der Verfügbarkeit
- Dass man sieht was passiert, mit graphischer Oberfläche
- Die Visualisierung der Programmieraufgaben
- -Allgemein: Gute Idee zum Üben der Diagramme -Keine Probleme mit der Speicherung oder der Ausführung

Antworten auf die Frage **„Was hat Ihnen nicht gefallen, was sollte geändert werden?“**:

- Wenn ich folgende drei Dinge nach dem Aufwand bewerten müsste:
  1. Aktivitätsdiagramm am PC erstellen (so war es gefordert) mit Code-Generierung
  2. Aktivitätsdiagramm auf Papier aufzeichnen
  3. Code ohne Aktivitätsdiagramm direkt schreiben
  so finde ich 1. am aufwendigsten. Insbesondere war die Bedienung von WARP mit der Maus nicht ganz einfach und das obwohl ich an einem Schreibtischcomputer gearbeitet habe. Die Leute mit Laptop müssen es da noch schwerer gehabt haben.
- Da man einen gewissen Prozentsatz zur Klausurzulassung benötigt, ist diese Aufgabe nicht sehr sinnvoll, denn man bekommt entweder alle Punkte oder gar keine.
- Viele technische Schwierigkeiten und Ausfälle (siehe Forum), Unmöglichkeit der Fehlerkorrektur (bei mir wurde aus unbekannten Grund eine überflüssige } gesetzt, der Fehler hielt sich auch nach erneutem Öffnen), insgesamt sinnvoller ist die direkte Eingabe von Code, schwierig Verbindungslinien zum Bearbeiten zu markieren, Verwendung von Flash.
- Ein wenig mehr Erklärungen wären super gewesen, aber an sich war das kein großer Makel

- Wenn man im nachhinein etwas hinzugefügt hat,was ausenrum um ein bestimmten block musste, konnte man den block nicht in das neue einfügen
- Die Hasenjagd hing stark vom startpunkt des hasen ab. Desweiteren wurden für verbesserungen punkte abgezogen
- Ungenaue Aufgabenstellungen, schwache technische Umsetzung der Tests. Ihr schafft es mal wieder die einzige Vorlesung zu sein, die ihre Übungsaufgaben nicht vernünftig hinbekommen, das ist demotivierend und auf Dauer ebenso nervtötend wie frustrierend...-.-
- war bei einigen aufgaben bis früh um 6 gesessen! genau wie jpp!
- -Teilweise Anzeigebugs am Cursor
  -Umständlich mehrere Aktivitäten zu kopieren, da dies nicht möglich war
  -Die Jägeraufgabe war zu „random“ und oft trotz gutem Code erst nach mehreren Versuchen lösbar
- Allerdings gab es noch viele Bugs, bzw kleine Details, welche für eine unkomfortable bzw erschwerte Bedienung sorgen.
- beim grafischen Durchlauf des Roboters Diagramm daneben anzeigen und stelle am Code markieren welche der Roboter grade ausführt
- Wenn es bei dem Übungsblatt darum ging, Aktivitätsdiagramme zu üben, dann halte ich eine direkte Code-Eingabe für wenig sinnvoll. Wenn es bei der Kaninchenjagd zu kompliziert ist, das nur über ein Aktivitätsdiagramm zu lösen, dann sollte man sich überlegen, ob man die Aufgabe ändern sollte. Außerdem fände ich es gut, wenn es eine Art Hilfeseite gäbe, wo man z.B. nachschauen kann, wie man die etwas komplizierteren Sachen wie den else-Zweig im if-Block erzeugt, weil man das nur mit viel Ausprobieren findet, da die Symbole echt klein sind.
- Meiner Meinung nach wäre es besser zuerst SWT als Vorlesung zu hören als ADS
- Die Fenstergrößenprobleme und der beschränkte zu verwendende Platz für die Diagramme sollten verbessert werden.
- Technik muss verlässlicher sein, Interfaces sollten intuitiver sein und alles auf HTML5 und nicht flash laufen
- - Die Bedienung des WARP ist zu aufwendig
  -man muss ich zu sehr mit dem Prozess des Codesns auseinander setzen bzw. zu weniger mit Aktivitätsdiagrammen an sich
  -Der Schwierigkeitsgrad der Aufgaben ist dem Inhalt der Vorlesung nicht angemessen
- hat nicht genau das übersetzt was es sollte....
- in der aktuellen Form ist das Programm eine Zumutung! Das Konzept ist gut, und es würde sicherlich auch Spaß machen wenn das Programm laufen würde! Aber es kann nicht sein, dass man einen Fernseher braucht um alles sehen zu können, dass bei mehrmaligen Erzeugen des JavaCodes immer unterschiedlicher

Code rauskommt und die Aufgabenstellung so unpräzise ist, dass richtige Lösungen 'falsch' sind! So kostet das Programm mehr Zeit und Nerven, als dass es hilft!

- Es gab so gut wie keine Einarbeitung, man saß somit alleine ca. 1 Stunde nur daran, überhaupt das Programm zu kapieren. Auf Papierform wäre das Ganze wesentlich einfacher gegangen. Ich finde, man sollte nicht für jedes Thema ein eigenes Programm erstellen, wo sich die Studenten jedesmal selbst einarbeiten müssen. Oftmals ist es Übertrieben das Ganze online zu machen.
- es tretten viele fehler auf, allerdings konnte ich in meinem fall gut workarounds finden.
- Java-Code sollte durch das Programm richtig und eindeutig erstellt werden und nicht bei wiederholter Ausführung anders generiert werden. Ausserdem sollte das Programm während dem Bearbeitungszeitraum immer funktionsfähig sein.
- unsinnige/falsche Fehlermeldungen + Fehlermeldungen die keinen Schluss darauf zulassen, wo der Fehler liegt. Schlechte Robustheit, Anzeigefehler. Das war zwar noch nicht alles aber ich hoffe das reicht.
- Kein Copy/Paste möglich. Um z.B einen If-Block in eine For-Schleife zu schieben, muss alles neu geschrieben werden. Kein Verschieben von Blöcken in andere Blöcke möglich.
- Der Zeitaufwand für Einarbeitung, langsame Eingabe der Aufgaben per Maus, lösen der Aufgaben und ausführliches Testen ist viel zu hoch und steht in keinem Verhältnis den Punkten die man für das Übungsblatt bekommt. Besonders in Anbetracht der komplexeren Aufgaben RandomHoover und Hasenjagd.
- Umständliche Bedienung, Langsam
- - Das Aufgabenblatt war sehr ungenau.
  - zu Anfang mit vielen Fehlern (die aber durch netten Support wieder behoben wurden)

  Zusatz: Noch bestehende auffällige Fehler im UML-Manager:
  - Wenn das Dropdown zum Ändern der Verbindungen(Pfeile oder Vererbungen) aufgerufen und dann neben dran geklickt wird, bleibt dieses im Bild und verschwindet nicht mehr
  - Es kann bei While-Schleifen-Blöcken (evt. auch bei den andern Blöcken) ein weiterer Block neben die if-Abfrage eingefügt werden (wenn der While-Block groß genug ist), dieser kann nicht mehr gelöscht werden und zerstört die While.
- Sehr viele Bugs am Anfang der Übung. Random generierter Code, Fehler bei Schleifen / Verzweigungen und so weiter.
  Sehr schlechte und ungenaue Aufgabenstellungen. Dadurch wusste man nie, welche Spezialfälle man behandeln musste und welche nicht. Außerdem waren die Rückgabe-Werte der Funktionen nicht beschrieben, wodurch man diese erraten musste.
  Ein wünschenswertes Feature wäre auch noch eine Art Debugger, damit man seine Fehler besser nachvollziehen kann.

- Buggs ohne Ende, dauernd überlastet/nicht erreichbar, ewige Fehlermeldungen die keinen Sinn ergeben/nichts mit dem Programmierten zutun haben wenn man den JavaCode ausführen will
- Eine WARP-bewertete Aufgabe reicht pro Übungsblatt.Der Rest sollte auch handschriftlich machbar sein, wie in anderen Vorlesungen auch (siehe Algorithmen & Datenstrukturen).
- Randomisierte Punkteverteilung im Fall der Kaninchenaufgabe. (Vorschlag: Jäger können sich nicht gegenseitig ausschalten, starten beide von der Mitte und das Kaninchen ist randomisiert irgendwo im Raum und bewegt sich zufällig) Teilweise zu wenig Informationen in der Fragestellung.
- Zu viele Bugs, überlastet bei zu vielen Nutzern, nicht auf Standart-Laptop Display ausgelegt
- - Eine Copy und Paste Funktion für Aktions- und Aktivitätsbausteine.
- Das Fenster in dem die Aktivitätsdiagramme erstellt werden sollte sich dynamisch an die Größe des Bildschirms anpasse. Das hast bei entsprechend großen Auflösungen (QHD und größer) sollte auch dieses Fenster größer dargestellt werden, sodass Scrollen innerhalb dieses Fensters nicht mehr oder zumindest seltener notwendig ist.
- Das Erstellen der Aktivitätsdiagramme lief aufs „Java in Kästchen schreiben“ hinaus, häufig benötigte Anweisungen sollten vorgefertigte Bausteine sein; Der Workflow war zu viel Trial and Error, insbesondere bei der Hasenjagd; Die Größe des Arbeitsbereichs sollte variabel sein
- Aufgaben genauer Stellen;
  Random-Code;
  Nicht Vollständig funktionierende Software;
- Noch bessere Lösung zum Bearbeiten auf kleineren Auflösungen bzw. z.B. Nebeneinanderanordnung von Graphik und Code
- Extrem umständliche Nutzung (Viele Funktionen nicht vorhanden, z.B. das Kopieren von Knoten oder das Verschieben derer in andere Aktivitäten)
  Extrem fehleranfällig & nicht verzeihend (z.B. sehr leicht im Browser vor/zurück zu springen. Dann werden gesamte Eingaben gelöscht)
  Problem Auflösung & Scrollen (anfangs unmöglich zu bedienen, dank Scrollbar verbessert)
  Problematische Java-Code Generierung (siehe z.T. Forum, z.B. werden leere Blöcke, Kommentare oder Methodennamen, teils auch if/else-statements fasch kompiliert)
  Schlechte Fehlerverfolgung (Bei mir kam es häufig vor, dass die Fehlermeldung lediglich sagt, dass ich das Diagramm einschicken sollte. Die Fehler waren z.T. jedoch banal wie eine Variable doppelt initialisiert.)
  Sehr wichtig: Nahezu unmöglich Logik-Fehler zu finden (Es gibt z.B. kein Debugger

wie in Eclipse um Variablenbelegung zu prüfen. Der Code kann überhaupt nicht nachverfolgt werden. Sehr schlecht, wenn man nicht versteht warum der Roboter in Richtung x abbiegt)
Schwierige Bedienbarkeit (z.B. Vergrößern von Feldern, Hinzufügen von Else-Blöcken, Einfügen neuer Aktivitäten oder Knoten in Blöcken: Jedes Mal muss ganz genau gezielt werden damit sich der Mauscurser verändert)
Darstellung schlecht (Normale Einstellung lassen die Verbindungen kreuz und quer durchs Diagramm laufen, über und unter den Knoten hinweg und deren Ankerpunkte an die Knoten können so knapp/schief liegen, dass man die Pfeile nicht mehr sehen kann. Ganz schlimm bei true/false-Pfaden, die man dann nicht mehr auseinander halten kann)
!Und vor allem! Nicht lösbar ohne Programmierkenntnisse (auch wenn dies in der Vorlesung so dargestellt wird, würde ich das ganz stark bezweifeln) Schon, dass Signaturen, Rückgabewerte und Parameter verstanden werden müssen, genauso wie boolesche Abfragen...
Soweit zum Programm (mehr Fehler siehe Forum, jetzt zu den Aufgaben)
Rechteck zu ungenau gestellt (manchmal hat es funktioniert, manchmal nicht, es gab andere Abfragen als nur ein 4x5 Rechteck zu Zeichen, denn das hatte ich jedes Mal. Außerdem war nicht angegeben ob gefüllt/ungefüllt)
Ähnlich Hoover-Random (wurde im Forum dann nachgetragen, dass Hindernisse nicht extrem gemein auftauchen können z.B. um 1 Feld völlig zu blockieren)
Hasenjagt (Auch wenn im Forum zuerst bestritten: Selbst der beste Jäger kann verlieren, denn wenn sich Hase+Gegner aufeinander zubewegen, verringert sich deren Weg in jedem Zug x2, während sich der eigene nicht verändert. Selbst eine Pause in jedem 3. Zug reicht da nicht.)

- sehr schwierig ohne ausreichend Programmierkenntnisse
  besser: abgestuft von sehr einfach bis sqqchwierig um sich mit dem Programm und Programmieren vertraut zu machen
- Sollte man einen äußeren Block bei einer Verschachtelung vergessen haben, so muss man alles noch einmal erstellen, genauso ist es mit komplexen Knoten, die sich bis auf kleine Details wiederholen. Es wäre einfacher, könnte man auch Aktionen aus einem Block in einen anderen ziehen, oder gesamte Blöcke, etc. kopieren, da es viel Zeit und Arbeit sparen würde.
- Der generierte Java-Code war für dasselbe Aktivitätsdiagramm bei zwei verschiedenen Benutzern ein deutlich anderer. (So wurde bei einer Person das Rechteck korrekt gezeichnet, bei der anderen wurden die Ecken nicht markiert).
  Ebenfalls war die Bewertung des Szenarios Rechtecks unverständlich, da einige korrekt gezeichnete Rechtecke nicht als korrekt akzeptiert wurden, obwohl sie vom Ablauf einigen bereits akzeptierten Durchläufen von Kommilitonen glichen.
- Das Programm, das wir für die Aufgaben verwendeteten war noch viel zu wenig ausgereift, so dass der Spass, den man haben könnte, da die Idee sehr schön ist,

zu Frust wurde. Viele logische Eingaben wurden nicht korrekt verarbeitet. Gerade bei Programmierung verlässt man sich aber auf die Logik. Ein Beispiel: ich habe in einer aktivität den rückgabetyp integer angegeben. die variable hieß meinentwegen „zahl“. dann hab ich einen aktion „return zahl“ geschrieben. Im Code stand dann einmal „return zahl“ und einmal „return res“ was überhaupt keinen Sinn ergibt und das Programm am funktionieren hinderte. „return res“ war nicht wegzubekommen, also hab ich mein return-statment gelöscht und die variable „zahl“ in „res“ umbenannt. Dann ging es. Allerdings hat das nur noch wenig mit Programmier Logik zu tun. Im Aktuellen Status ist daher das Programm eine Zumutung, die ewig viel Zeit und Nerven kostet. Man sollte es verwenden wenn es funktioniert.

- Die äußere Aktivität sollte man löschen können, ohne dass die inneren ebenfalls gelöscht werden.
  Aktivitäten, die z.B. in einem while-Block sind, sollten aus ihr heraus bewegt werden können.
- Java-Code wird nicht richtig erstellt. dadurch frust da die aufgabe richtig gelöst wurde, das diagramm aber nur nicht richtig umgesetzt wurde (if-else abfragen). Auserdem konnten die Aufgaben komplexer interpretiert werden, dadurch waren mehr funktionen im diagramm notwendig, diese konnten aber ab einer gewissen menge nichtmehr verarbeitet werden. auserdem fehlte copy and paste. Aufgaben zu Schwammig abgegrenzt zb hover-random
- Funktionsfähigkeit gleich null. Es war manchmal nicht erreichbar und hat nicht so funktioniert wie es sollte.
- Etliche Bugs
- - Man sollte Knoten ausschneiden/kopieren/einfügen können
  - Wenn der Browser aktualisiert / die Seite verlassen wird, sollte nicht ohne Warnung(!) alles gelöscht werden
  - Verschachtelte Schleifen sollten möglich sein
  - Buttons in if-Knoten für else-if-Verzweigung und else-Anweisung deutlicher hervorheben
  - System sollte während des Übungszeitraums verfügbar und aufrufbar sein
  - Fehler Pop-Ups, beispielsweise bei Endlosschleifen, sollten nicht den ganzen Bildschirm ausfüllen, sodass man sie nicht mehr wegklicken kann
  - Aufgabenstellung präziser formulieren (beispielsweise, dass bei Hoover erst der Knoten markiert werden muss, bevor sich der Roboter bewegen darf) oder größere Toleranz einbauen
- - extrem schlechte Bedienbarkeit
  - viele Fehler, die teilweise nicht direkt behoben werden konnten
  - ständiger Ausfall von den Servern
- man kann zB while-Schleifen nicht nachträglich in andere Strukturen verschachteln –> gesamtes Diagramm muss neu gemacht werden

- viele bugs
- -unklare Aufgabenstellung
  -verschachtelte Schleifen wurden nicht korrekt in java code compiliert gleiches für If Anweisungen
  -Copie und Paste war nicht verfügbar
  -zeitaufwand zum zeichen war ca. um den Faktor 10 höher als eigentlicher Programmieraufwand
- - Anfängliche technische Probleme
- Dadurch, dass viele der Aktivitätsdiagramme wegen unmöglicher Java-Umsetzung abgelehnt wurden, sollte man vllt überlegen, ob es so sinnvoll ist, weil am Ende sind viele der Aufgaben einfach als while-Schleifen programmiert worden, wo die Aktivitätsdiagramme mit den Pfeilen und alles ziemlich gering behandelt worden sind. Vllt wäre da eine Aufteilung in eine normale, schriftliche Abgabe und eine WARP-Abgabe sinnvoller, da ich zum Beispiel die ersten Aufgaben schön mit bedingten Abzweigungen gemacht habe, dies aber, dadurch, dass es mehrere waren, nicht angenommen wurden, obwohl sie vom Inhalt hundertprozentig richtig waren...
- Bedienbarkeit, z.B. kann man keine Aktivitäten verschieben solange der Cursor auf „Assoziation“ steht
- Fehlerkorrektur vor der eigentlichen Übungsaufgabe.
  Es gab eindeutig zu viele Probleme!
- -Einfügen von Do-While und Verzweigungen etwas umständlich
  -Bei Do-While-Schleifen usw. war es mir am Anfang nicht klar, dass man das Feld groß ziehen kann und neue Aktivitäten hinzufügen (ich dachte, es ginge nur eine)
  -Das Einfügen mehrerer Zeilen in eine Aktivität ist möglich (mit Semikolon) Ist das gewollt?
  -Beim Einfügen in Do-While-Schleifen usw. nervt es, dass diese immer noch ausgewählt sind und man sich so leicht beim Einfügen von Unterelementen verklickt
  -Kaninchen verhält sich manchmal sehr „einseitig“ und läuft einem der Jäger ohne eine Chance für den anderen vor die Flinte (s. meine Einsendung)

A.2.5.4. Umfrage-Ergebnisse zur Erstellung von Klassendiagrammen im Sommersemester 2014

Hinweis: Die folgenden Freitext-Anwtorten wurden unverändert übernommen, also inklusive Fehlern in Rechtschhreibung und Interpunktion.

Antworten auf die Frage **„Was hat Ihnen gut gefallen?“**:

- Schöne Anwendungsbeispiele

- Ich konnte noch einiges dazu lernen
- Gute umsetzung
- -Gute verständnisvolle Einführung zur Thematik. -Zeitsparender als auf einem Blatt Papier, da Änderungen desDiagramms leicht möglich sind.
- elektronische abgabe
- verständliche Fehlermeldung
- Ich finde gut, dass man sofort seine Bewertung bekommen hat, wenn man fertig war (bei den ersten beiden Aufgaben).
- Die CaseTrain Anwendung ist sehr gut. Es wird etwas praktisch angewandt und dadruch macht es Spaß etwas für SWT zu machen
- Mir hat gefallen, dass Aufgaben schnell umgesetzt bzw. bearbeitet werden konnten nachdem man eine Lösung gefunden hat.
- Vorallem in den Ersten 2 Casetrains wird man delaiiert durch den Konstruktionsprozess von Klassendiagrammen gefürht
- Gut war die zunehmende Schwierigkeit bei den CasteTrain-Aufgaben, sowie parallel der zunehmende Schwierigkeitsgrad im Programm durch selbstständiges Erstellen der Diagramme. Zudem war die Einarbeitung durch die Youtube-Videos angenehm.
- Aspekt der Interaktivität, gute Umsetzung des Konzepts.
- die steigende Komplexität der Aufgaben erleichtert das Verständnis und somit auch die Bearbeitung.
- nicht wirklich gut aber immerhin nutzbar
- + Schöne Art in Arbeit zu UML Klassendiagrammen einzusteigen
- Gutes Gesamtkonzept. Lösungen können nachträglich angezeigt werden.
- Das Prinzip
- Das Konzept und die damit verbundene leichte Korrektur sind optimal.
- Aufgaben online lösen; Lösungen nach jeder Aufgabe;
- leicht zu bedienen
- Einfach zu bedienen, fast keine Fehler
- Übersichtlichkeit am Computer besser als auf Papier bei Änderungen. Inhalt gut gewählt, da relativ leicht nachvollziehbar.
- Keine Fehler, sehr intuitiv bedienbar, alle mir bekannten SQL Befehle haben geklappt.
- war in ordnung aber auch nicht mehr
- Unumständliche Bearbeitung (keine Abgabe in Papierform) Ansprechendes Gesamtkonzept

- Gute Aufgaben, einfache Bedienung und in meinem Fall keine Probleme mit der Verfügbarkeit
- Die Visualisierung der Programmieraufgaben
- -Alle nötigen Funktionen vorhanden -Bedienung schnell verständlich

Antworten auf die Frage **„Was hat Ihnen nicht gefallen, was sollte geändert werden?“**:

- Verwendung von Flash, teilweise schwierig z.B. Verbindungslinien zum Bearbeiten zu markieren.
- Manche Aufgaben waren nicht lösbar, trotz Besuch der Vorlesungen und etc.
- Umständlich
- Korrekte Umsetzungen der Aufgabenstellung werden nicht anerkannt, UML-Diagramme sind nicht eindeutig korrekt lösbar,verschiedene Herangehensweisen an die Modellierung des Programms werden nicht berücksichtigt!
- allgemein, dass es deadlines gibt. ist ja schon ein jpp! hab die diagramme nicht mehr machen können weil ich dachte, dass die abgabe 23.55 uhr wie bei allen anderen kursen ist und ich mir extra den montag frei genommen hab.... bis ich bemerkte, dass die abgabe um 6 uhr rum war!
- -Die automatische Korrektur lässt manche Lösungen nicht zu, die prinzipiell auch korrekt währen.

  -Die (automatische?) Punkteeintragung in die Bewertung ist Fehlerhaft, da ich auf die Monopolyaufgabe laut Casetrainstatistik 80,2% habe, jedoch bei mir 0 Punkte eingetragen sind.
- wenn man bei einer assoziation auf das auswahlfenster klickt, kann es passieren, dass das drop-down-menü sich nicht mehr schließt (sogar, wenn man die assoziation löscht).
- Skalierung war sehr starr musste auf 1920x1080 viel scrollen
- Man sollte vielleicht klarer kenntlich machen, dass man nicht nur Klassen, sondern auch Enumerations etc. definieren kann, das entdeckt man nämlich eher durch Zufall.
- Meiner Meinung nach wäre es besser zuerst SWT als Vorlesung zu hören als ADS
- Nur eine vorgegebene Lösung ist richtig. Alternativlösungen wurden sofort als falsch bewertet.
- Eine automatische Auswrtung von Klassendiagrammen durch abgleichen mit "Mustern"halte ich nicht für sinnvoll. Sollte manuell geschehn durch Turoren etc.
- eine komplett elektronische Korrektur ist überhaupt nicht zweckmäßig, vor allem wenn dem Benutzer so viel Entscheidungsfreiraum gegeben wird

- Es wäre hilfreich gewesen, einen kompletten Fall in der Vorlesung oder im Tutorium zusammen zu bearbeiten...
- Ladezeiten, Robustheit des Systems, fummelige Bedienung.
- Aufgaben sollten eindeutiger deffiniert werden oder man sollte mehr freiheit bei der abgabe habe. Ein diagram sollte punkte erhalten auch wenn es nicht der musterlösung entspricht, aber schlüssig ist
- Unübersichtliche und wenig intuitive Bedienung. Sehr "fummelig", gilt auch für WARP
- ich halte die Korrektur der Klassendiagramme von Hand für sinnvoller.
- teilweise langsam
- - Das Hinzufügen der weiteren Elemente durch Fragen kann sehr verwirrend sein.
- Die UML-Diagramme sollten, meines Erachtens nach, besser per Hand korrigiert werden, da es nicht immer nur eine korrekte Lösung gibt. Außerdem würde man so auch ein persönliches Feedback bekommen, was dem Verstehen der Thematik zu Gute kommt.

  Teilweise ungenaue, bzw. unklare Aufgabenstellungen, die zu viel Spekulationen zulassen. Dadurch verliert man eventuell Punkte, obwohl man die Antwort genauso gut richtig sein könnte.
- Bei den Diagrammen nicht so strikt an eine Musterlösung halten und ggf von Hand korrigieren
- klarer machen, was gefordert ist, dh bei den offenen Klassendiagrammen denke ich, dass es nicht nur eine richtige Lösung gibt, aber man muss der Musterlösung entsprechen. Da entweder klarere richtlinien oder mehr Toleranz bei der bewertung
- Korrektur unbedingt von Hand machen, es gibt in UML nicht du DIE eine Lösung,alleine schon, kann der Computer nicht erkennen, wenn eine Methode anders benannt ist, als in der Musterlösung.
- Klassendiagramme sind niemals eindeutige Lösungen, sondern es gibt für ein Problem sehr viele optimale Lösungen die sich nur an unterschiedlichen Kapselungen unterscheiden. Dies ist dann natürlich nicht so optimal an einem Musterbeispiel die Lösungspunkte zu generieren.
- Anforderung der Aufgaben präziser formulieren; Die Größe des Arbeitsbereichs sollte variabel sein
- Korrektur von selbst erstellten UML-Klassendiagramm mittels Case Train. Lieber per Hand korriegieren lassen
- manchmal Fragestellungen nicht eindeutig
- Teilweise schwammige Aufgabenstellung.

- > Automatische Korrektur

  > Übungsfall sollte nicht bereits in der Vorlesung besprochen sein (Man konnte nicht üben, weil man Bibliothek schon kannte, die anderen aber nicht rückgängig machen konnte)

  > Im geführten Fall ist die Punktebalance sehr gefährlich (Wer ich bei den Klassen vertut, verliert gerne mal 30-40% des gesatem Falls)

  > @Monopoly: Alle Operationen kamen in eine Klasse, diese war dann jedoch so groß, dass sie von anderen Überdeckt wurde

  > Einige Fragen unklar (bei machen Kardinalitätsfragen stand nicht dabei wo das 1. Zeichen hinkommen wird)
- einzelne Klassen sollte man immer verschieben können(Beispiel Monopoly Klasse Spieler)
- Hinweise sollten automatisch eingeblendet werden, wie z.B. der Hinweis, wann Aufgaben endgültig eingetragen werden und nicht nur bei Klick auf den Ausrufezeichenbutton, da dieser von vielen übersehen wurde, aber doch hilfreiche Informationen enthält.
- Zur besseren Bedienbarkeit alles anklickbare deutlich sichtbar machen, z.B. Zahlen bei Beziehungen
- Sehr einfache Fragen die teilweise sehr oft wiederholt werden, bei denen nur die Beziehungen zu ändern sind
- - manche Aufgaben sind auslegungssache. vor allem bei komplexen aufgaben gibt es schonmal mehrere möglichkeiten.
- Funktionsfähigkeit gleich null
- - sich überlagernde Klassen beim Fall "Monopoly"

  - Assoziations-Pop-Up-Menüs, die sich nicht mehr schließen lassen
- - Aufgabenstellungen teilweise unklar (Anforderungen schwammig)

  - automatischer korrektur lässt nur eine Lösung zu
- dass das System keine Rückmeldung über Fehler gibt
- Fehlerkorrektur vor der eigentlichen Übungsaufgabe. Es gab eindeutig zu viele Probleme!
- -Elektronische Bewertung (Olympia) etwas unklar

  -Bei Monopoly sollte das zusätzliche Einfügen von Feldern usw. entfernt werden

  -Alle Methoden auf den Spieler abzuwälzen "passtirgendwie nicht so ganz - einige hätten, wie ich finde, besser zur Straße gepasst

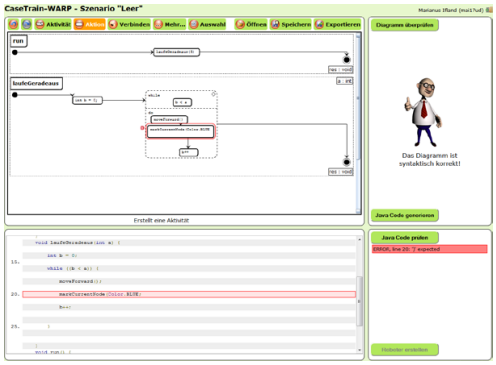
A.2.6. Sonstiges

A.2.6.1. CaseTrain-WARP Kurzanleitung

CaseTrain-WARP Kurzanleitung

In CaseTrain-WARP erstellen Benutzer Roboter durch die Eingabe von Aktivitätsdiagrammen. In der Editor-Komponente werden dabei eingegebene Diagramme automatisch in Java-Code übersetzt. Die Erstellung eines Roboters bzw. dessen Ausführung findet dabei immer im Kontext eines sogenannten Szenarios statt, welches die virtuelle Spielwelt definiert.

1. WARP-Editor



Die Editor-Komponente besteht aus vier Bereichen. Links oben ist der Editor, in dem Diagramme erstellt und auf lokalen Datenträger gespeichert bzw. von diesen geöffnet werden können. Rechts daneben befindet sich der Bereich für Fehlermeldungen bezüglich der syntaktischen Validität des Diagramms. Unter dem Editor befindet sich der Bereich für den generierten Java-Code. Rechts daneben befindet sich analog der Bereich für Fehlermeldungen bezüglich der syntaktischen Validität des Java-Codes. In beiden Bereichen wird beim Überfahren der Fehlermeldungen mit dem Mauszeiger die entsprechende Stelle im Diagramm (bzw. im Java-Code) farblich markiert. Die Schritte zur Erstellung eines Roboters sind: 1) Erstellen des Diagramms, 2) Prüfen des Diagramms, 3) automatisches Generieren von Java-Code 4), Überprüfen des Java-Codes 5), Erstellen des Roboters. Die Schritte 2-5 werden über entsprechende Buttons ausgelöst. Bei Erstellung eines Roboters wird ein sogenannter Roboter-Key erstellt, der den Roboter eindeutig identifiziert und über welchen er in der WARP-Arena betrachtet werden kann.

1.1 Generierung von Java-Code

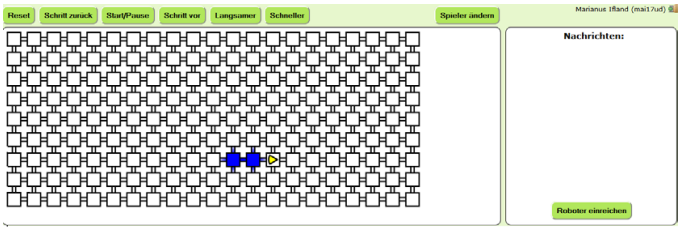
Bei der Generierung von Java-Code wird jede Aktivität des Diagramms in eine Methode der vorgegebenen Roboter-Klasse umgewandelt. Zur Definition von Rückgabetyp und Übergabeparametern der Aktivitäten muss eine entsprechende Methodensignatur in Java-Syntax angegeben werden. Methoden, deren Rückgabetyp nicht `void` ist, geben stets den Parameter `res` zurück. Dazu muss die Variable deklariert werden.

> Beispiel: Eine Aktivität mit dem Rückgabetype int muss einen Aktion beinhalten, in der die Variable `int res` deklariert wird.

Damit ein Roboter in der Arena betrachtet werden kann, muss er stets die Methode `run()` implementieren, es muss im Diagramm also eine Aktivität `run()` existieren.

Die WARP-Arena erreicht man u.a. durch den Button „Roboter ansehen“ in der Editor-Komponente, die erscheint, sobald ein Key erstellt wird.

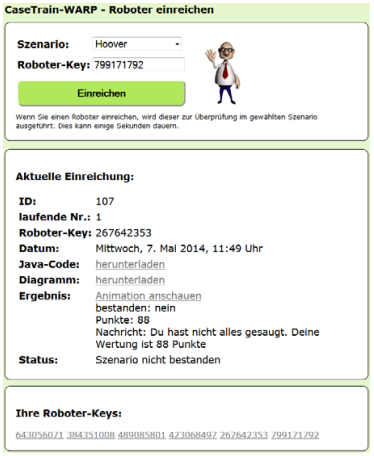
2. WARP-Arena



In der Arna können Sie nach Auswahl des Szenarios und Angabe des Roboter-Keys eine Animation erzeugen und betrachten, die das Verhalten des Roboters in der virtuellen Umgebung zeigt. Die Animation kann in der Geschwindigkeit variiert und auch schrittweise ausgeführt werden.

Die Arena dient zum ausführlichen Testen Ihrer Roboter, bevor Sie sie einreichen. Die Seite zur Einreichung erreicht man u.a. durch den Button „Roboter einreichen“ in der Arena-Komponente, die erscheint, sobald die Animation beendet wurde.

3. WARP-Einreichung



Nach Auswahl des Szenarios und Eingabe des Roboter-Keys können Sie einen Roboter einreichen. Dabei wird der Roboter im entsprechenden Szenario erneut ausgeführt und bewertet. Die entsprechende Animation kann anschließend betrachtet werden, ebenso können das zugehörige Aktivitätsdiagramm und der generierte Java-Code heruntergeladen werden. Eine Aufgabe gilt als bestanden, wenn der Status der aktuellen Einreichung „Szenario bestanden“ lautet.

Achtung: Eine Einreichung kann nicht rückgängig gemacht werden. Testen Sie Ihre Roboter vor einer Einreichung also ausgiebig in der Arena. Dies gilt insbesondere für Szenarien, die zufällige Elemente enthalten!

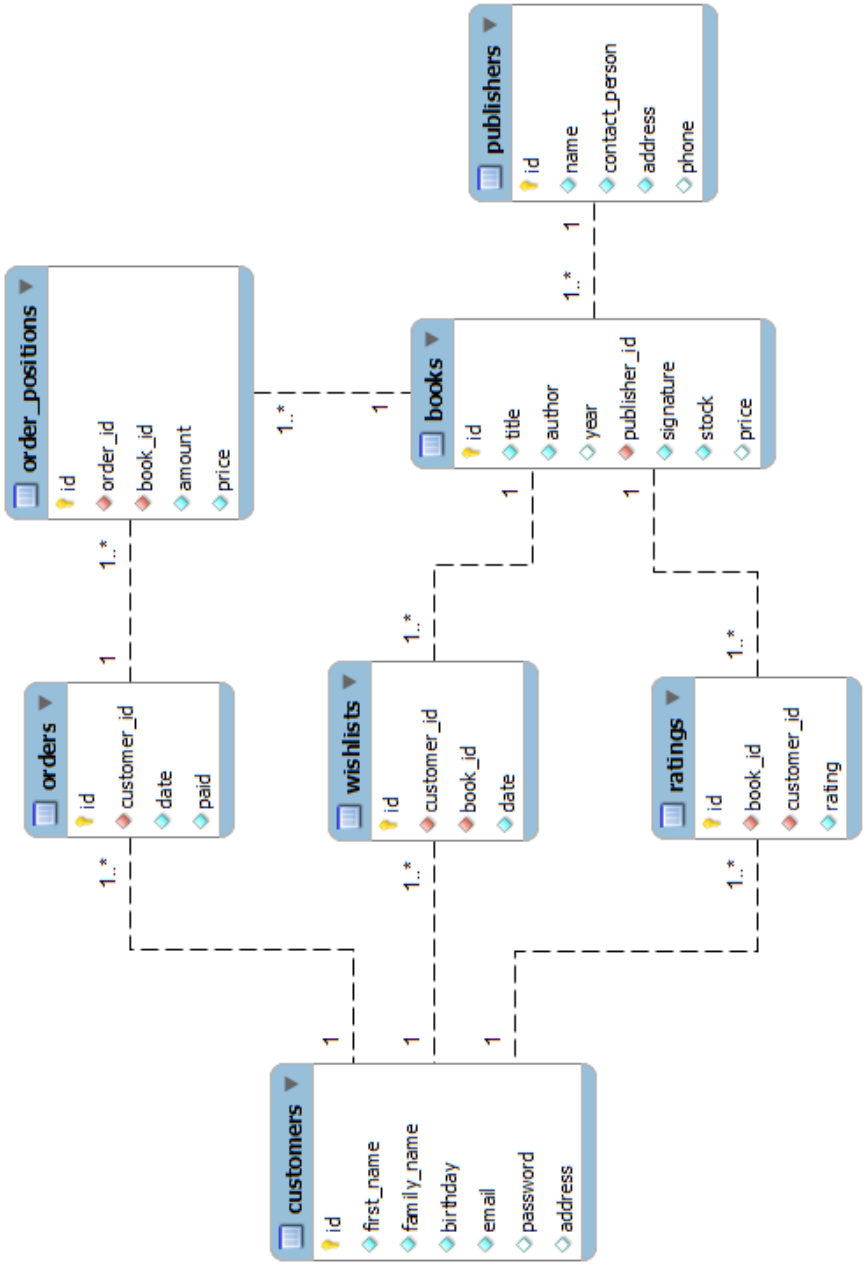
4. Screencasts

Es stehen zwei Screencasts zur Bedienung von WARP zu Verfügung:

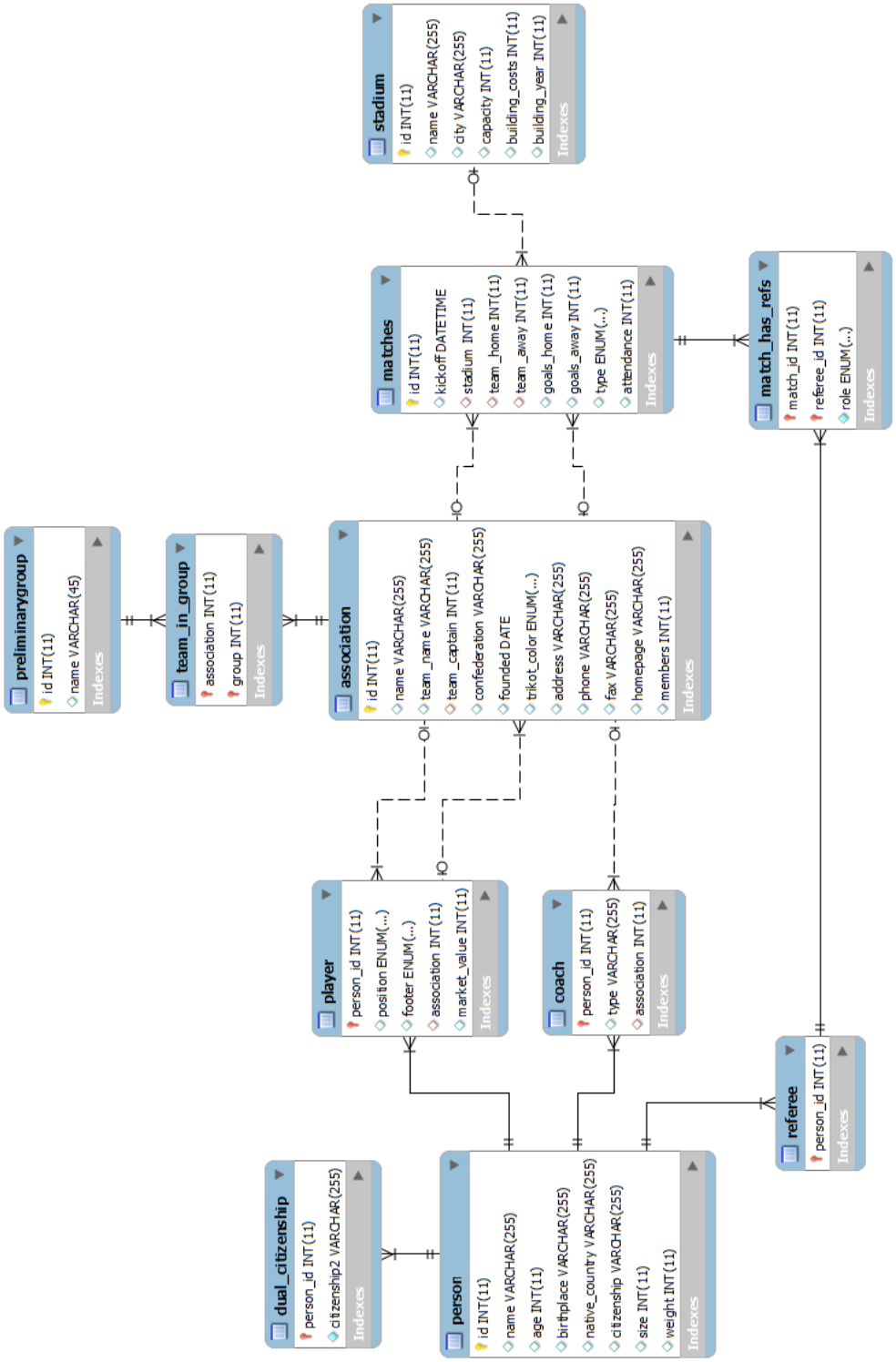
1. Aktivitätsdiagramm erstellen: http://youtu.be/y8pFpvt2pSE
2. Arena und Einreichung: http://youtu.be/mk5dyyNKTpE

A.2.6.2. ER-Diagramme der ÜPS Szenarios

Amazon Warenhaus:



Fußball Weltmeisterschaft 2010:



Literaturverzeichnis

[Ali et al., 2007] Ali, Noraida Haji, Z. Shukur und S. Idris (2007). *A design of an assessment system for UML class diagram*. In: *Proceedings - The 2007 International Conference on Computational Science and its Applications, ICCSA 2007*, S. 539–544.

[Anderson und Krathwohl, 2001] Anderson, Lorin W und D. R. Krathwohl (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman.

[Balzert, 2005] Balzert, H (2005). *Lehrbuch der Objektmodellierung: Analyse und Entwurf mit der UML 2*. Lehrbücher der Informatik. Elsevier.

[Bellon et al., 2007] Bellon, S., R. Koschke, G. Antoniol, J. Krinke und E. Merlo (2007). *Comparison and Evaluation of Clone Detection Tools*. IEEE Transactions on Software Engineering, 33(9):577–591.

[Bloom et al., 1956] Bloom, B S, M. Englehard, E. Furst, W. Hill und D. Krathwohl (1956). *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*, Bd. 16.

[Bloom, 1973] Bloom, Benjamin S. (1973). *Taxonomie von Lernzielen im kognitiven Bereich*. Beltz, Weinheim.

[Bransford et al., 1990] Bransford, John D, R. D. Sherwood, T. S. Hasselbring, C. K. Kinzer und S. M. Williams (1990). *Anchored instruction: Why we need it and how technology can help*. In: *Cognition education and multimedia Exploring ideas in high technology*, S. 115–141.

[Braschler und Ripplinger, 2003] Braschler, Martin und B. Ripplinger (2003). *Stemming and Decompounding for German Text Retrieval*. In: Sebastiani, Fabrizio, Hrsg.: *Advances in Information Retrieval*, Bd. 2633 d. Reihe *Lecture Notes in Computer Science*, S. 12. Springer Berlin / Heidelberg.

[Collins et al., 1989] Collins, Allan, J. S. Brown und S. E. Newman (1989). *Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics*. In: Resnick, Lauren B, Hrsg.: *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*, S. 453–494. Erlbaum.

[Cooper et al., 2000] Cooper, Stephen, W. Dann und R. Pausch (2000). *Alice: a 3-D tool for introductory programming concepts*. Journal of Computing Sciences in Colleges, 15:107–116.

[Eilers, 2006] Eilers, Björn (2006). *Entwicklung eines integrierbaren Systems zur computergestützten Lernfortschrittskontrolle im Hochschulumfeld*. In: Grob, Heinz Lothar und J. vom Brocke, Hrsg.: *Arbeitsberichte E-Learning*. European Research Center for Information Systems (ERCIS), Münster.

[Fincher et al., 2010] Fincher, Sally, S. Cooper, W. Lafayette, M. Kölling und J. Maloney (2010). *Comparing Alice , Greenfoot & Scratch*. In: *41st ACM technical symposium on Computer science education*, S. 192–193.

[Fowler und Beck, 2000] Fowler, Martin und K. Beck (2000). *Übel riechender Code*. In: *Refactoring oder: Wie Sie das Design vorhandener Software verbessern*, S. 67–82. Addison-Wesley.

[Gessenharter und Rauscher, 2011] Gessenharter, Dominik und M. Rauscher (2011). *Code Generation for UML 2 Activity Diagrams Towards a Comprehensive Model-Driven Development Approach*. In: *ECMFA'11 Proceedings of the 7th European conference on Modelling foundations and applications*, S. 205–220.

[Haladyna, 2004] Haladyna, T M (2004). *Developing and validating multiple-choice test items*, Bd. 31. Routledge; 3 edition (November 12, 2012).

[Hamp und Feldweg, 1997] Hamp, Birgit und H. Feldweg (1997). *GermaNet - a Lexical-Semantic Net for German*. In: *Proceedings of ACL workshop Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, S. 9–15.

[Hermann, 2013] Hermann, Felix (2013). *Erweiterung der Robotersimulationsumgebung RoSE auf Multiagentensysteme mit didaktischem Fokus*. Bachelorarbeit, Julius-Maximilians-Universität Würzburg.

[Hoffmann et al., 2008] Hoffmann, Andreas, A. Quast und R. Wismüller (2008). *Online-Übungssystem für die Programmierausbildung zur Einführung in die Informatik*. In: *DeLFI 2008: Die 6. e-Learning Fachtagung Informatik*, S. 173–184, Bonn.

[Hörnlein et al., 2009] Hörnlein, Alexander, M. Ifland, P. Klügl und F. Puppe (2009). *Konzeption und Evaluation eines fallbasierten Trainingssystems im universitätsweiten Einsatz (CaseTrain)*. GMS Medizinische Informatik, Biometrie und Epidemiologie, 5(1).

[Ifland et al., 2014a] Ifland, Marianus, F. Herrmann, J. Ott und F. Puppe (2014a). *WARP - ein Trainingssystem für UML-Aktivitätsdiagramme mit mehrschichtigem Feedback*. In: *DeLFI 2014 - Die 12. e-Learning Fachtagung Informatik*.

[Ifland et al., 2012] Ifland, Marianus, A. Hörnlein, J. Ott und F. Puppe (2012). *Geführtes Tutorsystem mit Unterstützung zunehmender Selbstständigkeit zur Modellierung von UML-Klassendiagrammen*. In: Desel, Jörg, J. M. Haake und C. Spannagel, Hrsg.: *DeLFI 2012 - Die 10. e-Learning Fachtagung Informatik*, S. 75–86, Hagen und Heidelberg. Gesellschaft für Informatik.

[Ifland et al., 2014b] Ifland, Marianus, M. Jedich, C. Schneider und F. Puppe (2014b). *ÜPS - Ein autorenfreundliches Trainingssystem für SQL-Anfragen*. In: *DeLFI 2014 - Die 12. e-Learning Fachtagung Informatik*.

[Ifland et al., 2010] Ifland, Marianus, J. Ott, A. Hörnlein und F. Puppe (2010). *Integration eines Freihandzeichen-Tools in das Trainings- und Prüfungssystem CaseTrain*. Poster. präsentiert auf dem 14. Workshop der gmds-Arbeitsgruppe „Computerunterstützte Lehr- und Lernsysteme in der Medizin (CBT)“ und des GMA-Ausschusses „Neue Medien“.

[Koelling, 2010] Koelling, Michael (2010). *The Greenfoot Programming Environment*. ACM Transactions on Computing Education (TOCE), 10(4).

[Körndle et al., 2004] Körndle, Hermann, S. Narciss und A. Proske (2004). *Konstruktion interaktiver Lernaufgaben für die universitäre Lehre*. In: Carstensen, Doris und B. Barrios, Hrsg.: *Campus 2004 - Kommen die digitalen Medien an den Hochschulen in die Jahre?*, S. 57–67. Waxmann.

[Krinke et al., 2002] Krinke, Jens, M. Störzer und A. Zeller (2002). *Webbasierte Programmierpraktika mit Praktomat*. Softwaretechnik-Trends, 22(3):51–53.

[Kuhn, 1955] Kuhn, H. W. (1955). *The Hungarian method for the assignment problem*. Naval Research Logistics Quarterly, 2(1-2):83–97.

[Levenshtein, 1966] Levenshtein, V I (1966). *Binary Codes Capable of Correcting Deletions, Insertions, and Reversals*. Soviet Physics Doklady, 10:707–710.

[Lienert und Raatz, 1998] Lienert, Gustav A. und G. Raatz (1998). *Die Testaufgabentypen*. In: *Testaufbau und Testanalyse*, Kap. 1.4, S. 18–23. Beltz.

[Lovins, 1968] Lovins, Julie Beth (1968). *Development of a stemming algorithm*. Mechanical Translation and Computational Linguistics, 11(June):22–31.

[Maloney et al., 2010] Maloney, John, M. Resnick, N. Rusk, B. Silverman und E. Eastmond (2010). *The Scratch Programming Language and Environment*. ACM Transactions on Computing Education (TOCE), 10:16:1–16:15.

[Miller, 1956] Miller, George A (1956). *The magical number seven, plus or minus two: some limits on our capacity for processing information*. Psychological Review, 63(2):81–97.

[Mitrovic, 2003] Mitrovic, Antonija (2003). *An Intelligent SQL Tutor on the Web*. International Journal of Artificial Intelligence in Education, 13(2-4):173–197.

[Moulinier et al., 2001] Moulinier, Isabelle, J. McCulloh und E. Lund (2001). *West Group at CLEF 2000: Non-english Monolingual Retrieval*. In: Peters, Carol, Hrsg.: *Cross-Language Information Retrieval and Evaluation*, Bd. 2069 d. Reihe *Lecture Notes in Computer Science*, S. 253–260. Springer Berlin Heidelberg.

[Myers, 1990] Myers, Brad A. (1990). *Taxonomies of visual programming and program visualization*.

[Narciss, 2006] Narciss, Susanne (2006). *Informatives tutorielles Feedback: Ableitung und empirische Überprüfung von Entwicklungs- und Evaluationsprinzipien auf der Basis instruktionspsychologischer Erkenntnisse*. Waxmann, Münster.

[Nickel et al., 2000] Nickel, U., J. Niere und A. Zundorf (2000). *The FUJABA environment*. Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium.

[Niegemann et al., 2008] Niegemann, Helmut M., S. Domagk, S. Hessel, A. Hein, M. Hupfer und A. Zobel (2008). *Kompendium multimediales Lernen*. Springer.

[Ott, 2011] Ott, Julian (2011). *CaseTrain Paditor Framework - Diagrammbasierte Aufgabentypen in fallbasierten Trainingssystemen*. Bachelorarbeit, Julius-Maximilians-Universität Würzburg.

[Pahl und Kenny, 2009] Pahl, C. und C. Kenny (2009). *Interactive Correction and Recommendation for Computer Language Learning and Training*. IEEE Transactions on Knowledge and Data Engineering, 21(6):854–866.

[Porter, 2001] Porter, Martin (2001). *Snowball: A language for stemming algorithms*.

[Reif, 1995] Reif, Frederick (1995). *Understanding Basic Mechanics*. John Wiley & Sons.

[Roy und Cordy, 2007] Roy, Chanchal K und J. R. Cordy (2007). *A Survey on Software Clone Detection Research*. Queen's School of Computing TR, 115:115.

[Rumbaugh et al., 2010] Rumbaugh, J, I. Jacobson und G. Booch (2010). *The Unified Modeling Language Reference Manual, (Paperback)*. The Addison-Wesley object technology series. ADDISON WESLEY Publishing Company Incorporated.

[Rütter, 1982] Rütter, Theodor (1982). *Formen der Testaufgabe*. In: Klauer, Karl Josef, Hrsg.: *Handbuch der pädagogischen Diagnostik, Band I*, Kap. 3, S. 257–280. Urban Lissmann.

[Schmees et al., 2013] Schmees, Markus, M. Krüger und E. Schaper (2013). *E-Assessments in der Lehre: Ansätze , Nutzen & Technologien*. In: Schmees, Markus und M. Krüger, Hrsg.: *E-Assessments in der Hochschullehre: Einführung, Positionen & Einsatzbeispiele*, S. 19–32. Peter Lang, Frankfurt.

[Schunk, 2002] Schunk, Gunther (2002). *Studienbuch zur Einführung in die deutsche Sprachwissenschaft*. Königshausen & Neumann GmbH, Würzburg.

[Soler et al., 2010] Soler, Josep, I. Boada, F. Prados, J. Poch und R. Fabregat (2010). *A web-based e-learning tool for UML class diagrams*. In: *IEEE EDUCON 2010 Conference*, S. 973–979. Ieee.

[Soler et al., 2006] Soler, Josep, F. Prados, I. Boada und J. Poch (2006). *A Web-based tool for teaching and learning SQL*. In: *International Conference on Information Technology Based Higher Education and Training, ITHET*.

[Striewe und Goedicke, 2011] Striewe, Michael und M. Goedicke (2011). *Automated checks on UML diagrams*. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*, S. 38, New York, USA. ACM Press.

[Tholander und Karlgren, 2001] Tholander, Jakob und K. Karlgren (2001). *Support for Cognitive Apprenticeship in Object-Oriented Model Construction*. In: *Proceedings of CSCL2002*.

[Ufert et al., 2013] Ufert, Anja, A. Grillenberger und T. Brinda (2013). *eledSQL Entwicklung und Erprobung einer webbasierten Lernumgebung für Datenbanken und SQL*. In: Breiter, Andreas und C. Rensing, Hrsg.: *DeLFI 2013 - Die 11. e-Learning Fachtagung Informatik*, S. 167–178, Bonn.

[Usman und Nadeem, 2009] Usman, Muhammad und A. Nadeem (2009). *Automatic Generation of Java Code from UML Diagrams using UJECTOR*. International Journal of Software Engineering and Its Applications, 3(2):21–38.

[Vogt und Schneider, 2009] Vogt, Michael und S. Schneider (2009). *E-Klausuren an Hochschulen : Didaktik - Technik - Systeme - Recht - Praxis*.

[Wagner et al., 2006] Wagner, Richard, D. Zenker, C. Schäfer und S. Schneider (2006). *k-MED - vom lokalen Projekt zum zum e-Learning-Dienstleister*. GMS Medizinische Informatik, Biometrie und Epidemiologie, 2(3):1–8.

[Wolber und Street, 2011] Wolber, David und F. Street (2011). *App Inventor and Real-World Motivation*. In: *Proceedings of the 42nd ACM technical symposium on Computer science education*, S. 601–606.