Karlsruher Institut für Technologie

Ein Datensatz handgezeichneter UML-Klassendiagramme für maschinelle Lernverfahren

Bachelorarbeit von

Philipp Nicolas Schumacher

an der Fakultät für Informatik
KASTEL – Institut für Informationssicherheit und Verlässlichkeit

Erstgutachter:	Prof. Dr.-Ing. Anne Koziolek
Zweitgutachter:	Prof. Dr. Ralf Reussner
Betreuender Mitarbeiter:	Dominik Fuchß, M.Sc.
Zweiter betreuender Mitarbeiter:	Sophie Schulz, M.Sc.

23. August 2021 – 23. Dezember 2021

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 23. Dezember 2021

. . . . . .

(Philipp Nicolas Schumacher)

Zusammenfassung

Klassendiagramme ermöglichen die grafische Modellierung eines Softwaresystems. Insbesondere zu Beginn von Softwareprojekten entstehen diese als handgezeichnete Skizzen auf nicht-digitalen Eingabegeräten wie Papier oder Whiteboards. Das Festhalten von Skizzen dieser Art ist folglich auf eine fotografische Lösung beschränkt. Eine digitale Weiterverarbeitung einer auf einem Bild gesicherten Klassendiagrammskizze ist ohne manuelle Rekonstruktion in ein maschinell verarbeitbares Diagramm nicht möglich.

Maschinelle Lernverfahren können durch eine Skizzenerkennung eine automatisierte Transformation in ein digitales Modell gewährleisten. Voraussetzung für diese Verfahren sind annotierte Trainingsdaten. Für UML-Klassendiagramme sind solche bislang nicht veröffentlicht.

Diese Arbeit beschäftigt sich mit der Erstellung eines Datensatzes annotierter UML-Klassendiagrammskizzen für maschinelle Lernverfahren. Hierfür wird eine Datenerhebung, ein Werkzeug für das Annotieren von UML-Klassendiagrammen und eine Konvertierung der Daten in ein Eingabeformat für das maschinelle Lernen präsentiert. Der annotierte Datensatz wird im Anschluss anhand seiner Vielfältigkeit, Detailtiefe und Größe bewertet. Zur weiteren Evaluation wird der Einsatz des Datensatzes an einem maschinellen Lernverfahren validiert. Das Lernverfahren ist nach dem Training der Daten in der Lage, Knoten mit einem F_1-Maß von über 99%, Textpositionen mit einem F_1-Maß von über 87% und Kanten mit einem F_1-Maß von über 71% zu erkennen. Die Evaluation zeigt folglich, dass sich der Datensatz für den Einsatz maschineller Lernverfahren eignet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Motivation

Die grafische Modellierung eines Softwaresystems ist ein heutzutage weit verbreiteter Prozess [1], welcher der tatsächlichen Entwicklung von Software vorausgehen kann. Ein solcher Vorgang kann unter anderem das gemeinsame Systemverständnis innerhalb eines Teams [1, 2] oder auch die Wartbarkeit einer Software [3] stärken. Des Weiteren kann eine umfangreiche Modellierung dem Risiko einer schlechten Produktqualität oder Missverständnissen mit Stakeholdern entgegenwirken [2]. Industriestandard für das Modellieren von Software ist die *Unified Modeling Language* (UML) [4], eine weit verbreitete und anerkannte Modellierungssprache [5]. Diese ermöglicht mit verschiedensten Diagrammtypen die grafische Darstellung von Anforderungen, Prozessen und Softwaresystemen. Zu diesen Typen zählen auch die *UML-Klassendiagramme*, welche in der Praxis als eine der wichtigsten und am häufigsten eingesetzten UML-Diagrammarten gelten [6, 7].

Das Modellieren dieser Diagramme wird von zahlreichen Werkzeugen unterstützt. Sogenannte *Computer-Aided Software Engineering (CASE)-Tools*, wie *Enterprise Architect* [8], *StarUML* [9], *Microsoft Visio* [10] oder *PlantUML* [11], ermöglichen dabei die Modellierung und das Speichern von UML-Klassendiagrammen in einem digitalen Format. Spätere Änderungen an Diagrammen, die auf diese Art und Weise entstanden sind, lassen sich so sehr leicht und schnell tätigen, da man jederzeit auf die Ursprungsform des Klassendiagramms zugreifen kann. Der bei CASE-Tools bestehende Zwang an feste Regeln der Diagrammsyntax erschwert jedoch die freie Systemdarstellung und mindert das damit entstehende Verständnis [6]. Systemarchitekten halten sich in der Praxis aus diesem Grund eher lose an die exakte Spezifikation der UML [2]. In frühen Phasen eines Softwareprojektes entstehen Diagramme auch deshalb initial meist händisch auf nicht-digitalen Medien wie Papier oder einem Whiteboard [12]. Möchte man ein handgezeichnetes UML-Klassendiagramm (siehe Abbildung 1.1) festhalten, so beschränkt sich die Speicherung des Diagramms auf eine fotografische Lösung. Dies wiederum hat zur Folge, dass sich Änderungen am Diagramm nicht ohne Weiteres durchführen lassen wie noch mit von CASE-Tools erstellten Klassendiagrammen.

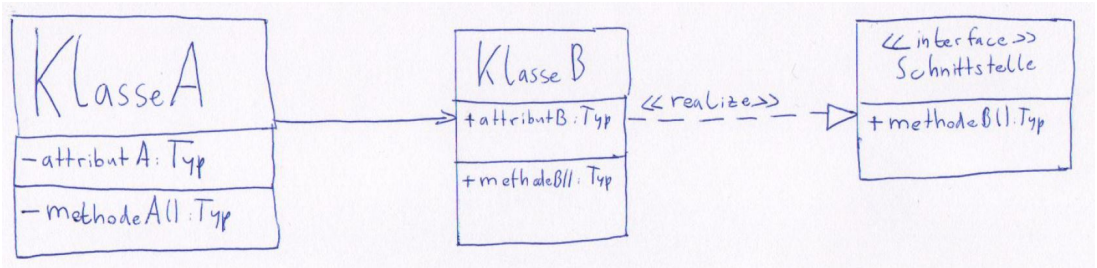


Abbildung 1.1.: Beispiel einer UML-Klassendiagrammskizze

Eine mögliche Lösung hierfür ist eine einmalige Rekonstruktion der Skizze in ein maschinell verarbeitbares Format, auf dem zukünftige Änderungen getätigt werden können. Dieser Schritt müsste derzeit vollständig manuell abgehandelt werden und wäre daher, insbesondere bei größeren Diagrammen, mit einem zusätzlichen Zeitaufwand verbunden. Eine Automatisierung dieses Prozesses ist daher wünschenswert.

Verwandte Ansätze der Skizzenerkennung anderer Diagrammdomänen [13, 14, 15, 16] haben in den letzten Jahren vermehrt gezeigt, dass maschinelle Lernverfahren diese automatische Transformation von Skizze in ein maschinell verarbeitbares Format gewährleisten können. Voraussetzung hierfür sind vollständig annotierte Diagrammbilder, auf denen sowohl die Position als auch die Größe alle der darin enthaltenen Elemente mit ihrer zugehörigen Kategorie klassifiziert wurden. Diese Daten können dem Training maschineller Lernverfahren dienen und somit die spätere Erkennung von Diagrammelementen ermöglichen (siehe Abbildung 1.2). Für handgezeichnete UML-Klassendiagramme sind Daten, welche auf diese Art und Weise aufbereitet wurden, noch nicht öffentlich zugänglich. Der Einsatz von maschinellen Lernverfahren ist in der Domäne handgezeichneter UML-Klassendiagramme daher noch nicht darstellbar.

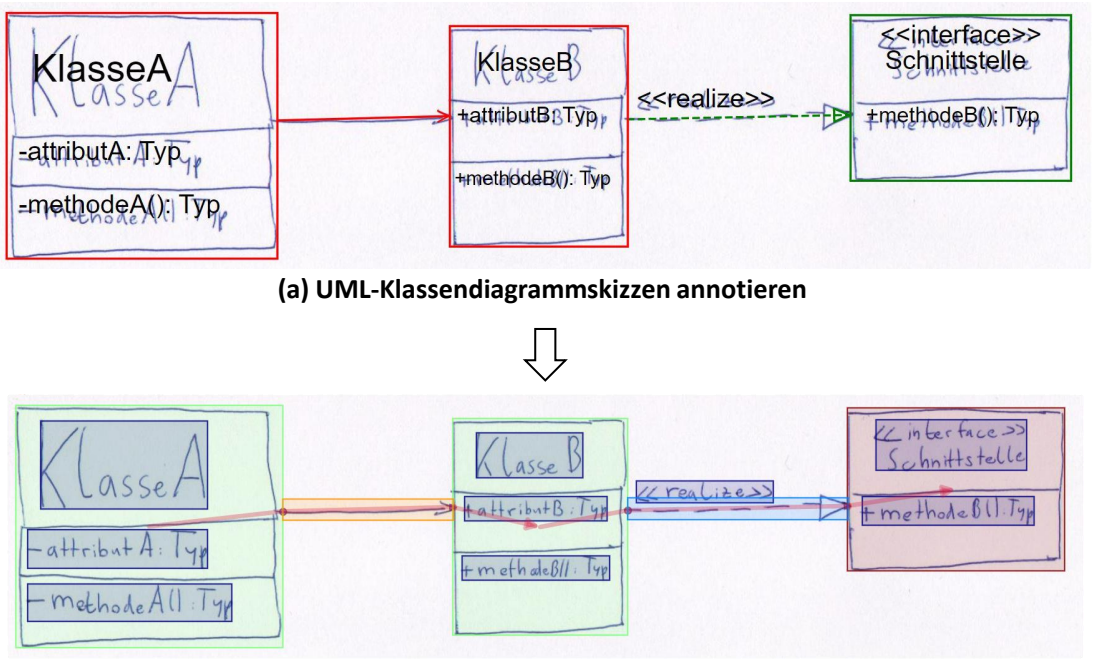


Abbildung 1.2.: **(a)** Annotation und **(b)** anvisierte Erkennung eines handgezeichneten UML-Klassendiagramms

Um diesem Problem entgegen zu wirken, stellt diese Bachelorarbeit einen Datensatz vollständig annotierter UML-Klassendiagrammskizzen vor. Dieser soll zukünftig den Einsatz maschineller Lernverfahren im Bereich handgezeichneter UML-Klassendiagramme ermöglichen und somit eine Erkennung dieser unterstützen. Damit einhergehend stellt die

Arbeit eine Datenerhebung von möglichst realitätsnahen UML-Klassendiagrammskizzen vor und zeigt anschließend deren Eigenschaften auf.

Des Weiteren werden Softwarewerkzeuge für das Annotieren und Konvertieren von UML-Klassendiagrammskizzen präsentiert, welche die Verwendung des Datensatzes für maschinelle Lernverfahren gewährleisten sollen. Dabei wird explizit der Einsatz dieser Werkzeuge auf denen im Zuge dieser Arbeit gewonnenen Daten detailliert beschrieben. Der Datensatz wird anschließend initial für ein bereits existierendes, maschinelles Lernverfahren eingesetzt und getestet. Darauffolgend wird der Datensatz auf Grundlage der hierbei entstehenden Ergebnisse und weiterer Metriken evaluiert.

Die Arbeit ist wie folgt strukturiert: Kapitel 2 führt benötigte Grundlagen ein. Hierbei werden sowohl für die Arbeit relevante Elemente von UML-Klassendiagrammen präsentiert, als auch Grundlagen des maschinellen Lernens eingeführt, welche den Aufbau des Datensatzes beinflussen. Kapitel 3 stellt im Anschluss verwandte Arbeiten und Datensätze aus dem Bereich der Skizzenerkennung und der UML vor.

Kapitel 4 beschäftigt sich mit der Erhebung von UML-Klassendiagrammskizzen. Dabei werden die Quellen der Datengewinnung präsentiert, sowie Charakteristiken der so gewonnenen Daten beschrieben. In Folge dessen wird in Kapitel 5 das für das Annotieren genutzte Werkzeug vorgestellt und die damit durchgeführte Annotation näher erläutert. Zusätzlich wird eine Transformation der Daten in ein für ein maschinelles Lernverfahren nutzbares Format beschrieben.

Kapitel 6 evaluiert schließlich den Einsatz des Datensatzes für ein maschinelles Lernverfahren. In diesem Schritt werden die zugehörigen Ergebnisse analysiert und eingeordnet, um den Datensatz auf seine Validität zu prüfen. Des Weiteren wird dieser anhand seiner Vielfältigkeit, Detailtiefe und Größe bewertet. Kapitel 7 fasst schließlich die Arbeit zusammen und beschreibt, wie der Datensatz in zukünftigen Arbeiten verbessert und erweitert werden kann.

2. Grundlagen

Im Folgenden werden die für diese Arbeit benötigten Grundlagen eingeführt. Abschnitt 2.1 beschäftigt sich in einem ersten Schritt mit UML-Klassendiagrammen. Hierbei werden UML-Elementtypen vorgestellt, welche für den späteren Verlauf der Arbeit von Relevanz sind. Des Weiteren werden Besonderheiten und Herausforderungen der Domäne von Diagrammskizzen in Bezug auf das Erstellen eines Datensatzes und auf das darauf aufbauende maschinelle Lernen dargestellt. Abschnitt 2.2 führt daraufhin Grundlagen des Einsatzes von maschinellen Lernverfahren im Bereich der Skizzenerkennung ein und stellt schließlich dar, wie diese den Aufbau des Datensatzes beinflussen. Abschließend wird in Abschnitt 2.3 der *Common Objects in Context* (COCO)-Datensatz [17] und das zugrundeliegende Datenformat [18] vorgestellt, welches das erwartete Eingabeformat für eine Vielzahl von maschinellen Lernverfahren ist. Dabei wird der Aufbau dieses Formats detailliert beschrieben und auf den UML-Klassendiagrammkontext übertragen.

2.1. UML-Klassendiagramme

Die Unified Modelling Language (UML) [4] ist eine grafische Modellierungssprache, die im Jahre 1997 von der *Object Management Group* (OMG) entwickelt wurde. Sie umfasst dabei 13 verschiedene Diagrammtypen, die in jeweils unterschiedlichen Anwendungsbereichen zum Einsatz kommen können. *UML-Aktivitätsdiagramme* oder *UML-Zustandsdiagramme* modellieren so beispielsweise das Verhalten eines Softwaresystems, während andere Diagramme dessen statische Struktur beschreiben. Zum Einsatz für Letzteres kommen vor allem UML-Klassendiagramme. Diese können dabei sowohl von zahlreichen Computer-Aided Software Engineering (CASE)-Tools [8, 9, 10, 11] generiert werden, als auch händisch auf einem nicht-digitalen Medium wie Papier oder einem Whiteboard entstehen. Wie in Kapitel 1 bereits dargestellt, werden nachfolgend vor allem Skizzen, also handgezeichnete Klassendiagramme, Teil der Betrachtung werden.

Im Folgenden werden in Unterabschnitt 2.1.1 für den Datensatz relevante Elementkategorien der UML-Klassendiagrammsyntax vorgestellt. Dabei wird Bezug auf die geometrische Form der jeweiligen Elementtypen genommen, da diese für eine Diagrammerkennung von besonders hoher Relevanz ist. Darauffolgend werden in Unterabschnitt 2.1.2 Herausforderungen und Probleme illustriert, welche speziell bei handgezeichneten Klassendiagrammen auftreten können. Hierbei wird ebenso verdeutlicht, inwiefern sich Diagrammskizzen von digital erzeugten Diagrammen unterscheiden und wieso eine Abgrenzung von diesen beim Erstellen eines Datensatzes sinnvoll ist.

2.1.1. Elementkategorien

Die in dieser Arbeit referenzierte UML-Klassendiagrammsyntax bezieht sich auf die offizielle UML 2.5 Spezifikation der OMG [4]. Diese umfasst eine Menge von Elementkategorien, welche sich wiederum in Untermengen von *Knoten* und *Kanten* einordnen lässt. Beim Trainieren maschineller Objektdetektoren ist dabei in erster Linie deren geometrische Form für die Klassifikation ausschlaggebend. Dementsprechend werden im Folgenden alle vorkommenden Elemente mit dieser präsentiert. Zusätzlich ist Text ein elementarer Bestandteil von UML-Klassendiagrammen, welcher sowohl Knoten als auch Kanten beschriften kann. Alle existierenden Textkategorien werden daher ebenfalls vorgestellt.

Knoten Die in Klassendiagrammen vorkommenden Knoten symbolisieren zumeist Softwarebestandteile. Hierzu gehören unter anderem *Klassen*, *abstrakte Klassen*, *Schnittstellen*, *Enumerationen* und *Objekte*. Diese werden in der UML als eine Zusammensetzung von teilweise mehreren Rechtecken visualisiert und können dabei jeweils verschiedene Arten von Textblöcken enthalten (siehe Abbildung 2.1).

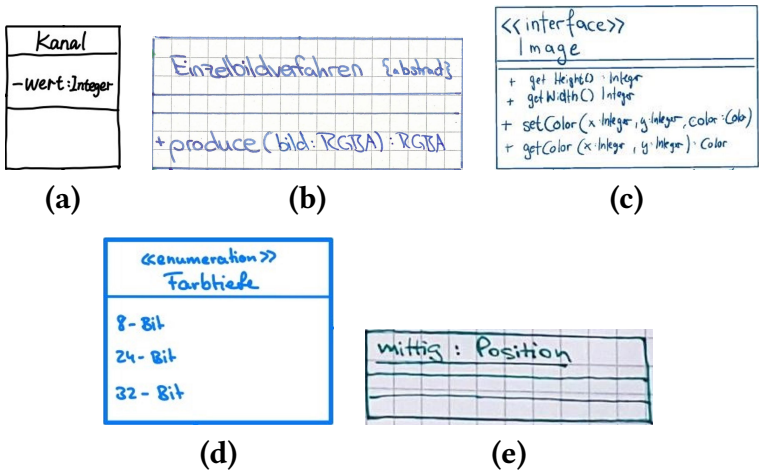


Abbildung 2.1.: **(a)** Klasse, **(b)** abstrakte Klasse, **(c)** Schnittstelle, **(d)** Enumeration und **(e)** Objekt aus jeweils verschiedenen Skizzen des in dieser Arbeit präsentierten Datensatzes

Zu diesen Textkategorien zählen sowohl der *Elementname* als auch *Attribute* und *Methoden*. Die zwei letztgenannten bestehen dabei in der Regel aus einer *Sichtbarkeit*, einem *Attribut- oder Methodennamen* und einem *Rückgabetypen*. Text ist ebenfalls für die Separation der genannten Elemente essenziell, da sich ihre geometrische Form im Grunde nicht unterscheidet. Abstrakte Klassen können so beispielsweise durch eine kursive Schriftart des Klassennamens oder durch den Zusatz *{abstract}* ausgezeichnet werden, während Schnittstellen vor ihrem Namen in der Regel die Textzeile *«interface»* besitzen. Gleiches gilt für Enumerationen, denen analog zu Schnittstellen die Textbeschriftung *«enumeration»* vorhergeht. Objekte hingegen sind durch einen unterstrichenen Namen oder den durch einen Doppelpunkt verbundenen Zusatz des zugehörigen Klassennamens gekennzeichnet.

Von CASE-Tools generierte Diagramme können von den genannten Definitionen zur visuellen Trennung dieser Elemente teilweise abweichen (siehe Abbildung 2.2).

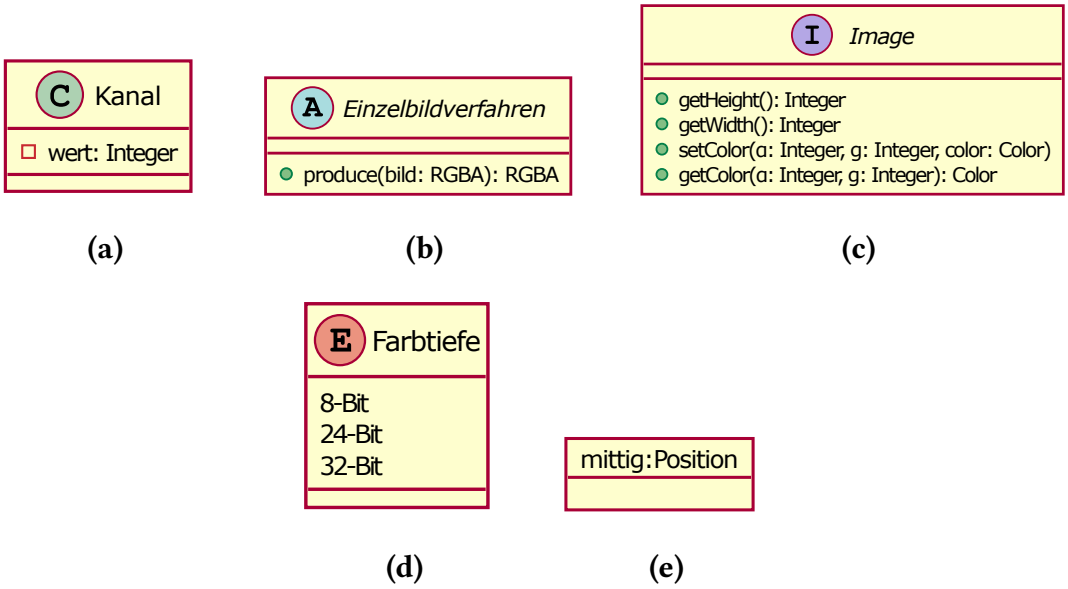
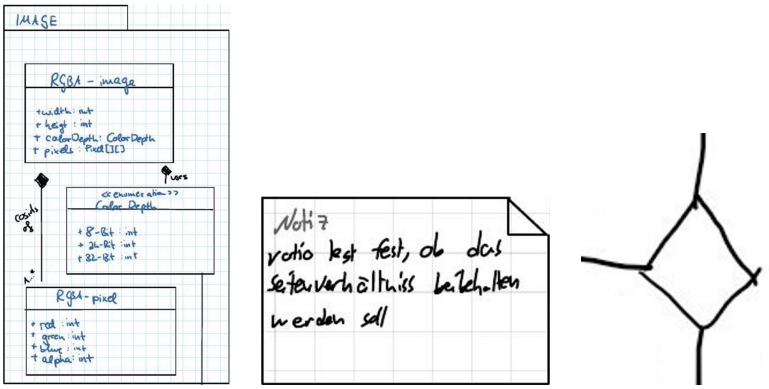


Abbildung 2.2.: Knoten aus Abbildung 2.1, erzeugt mit dem CASE-Tool PlantUML [11]. Die Separation und Klassifikation der Elemente erfolgt in der Regel über ein sich links neben dem Namen befindenden Symbol.

Eine Sonderform von Knoten nehmen *Pakete* in UML-Klassendiagrammen ein (siehe Abbildung 2.3). Diese repräsentieren ebenfalls einen Softwarebestandteil und können als einziger Knoten weitere Elemente in sich tragen. Visualisiert werden sie durch eine Zusammensetzung zweier Rechtecke und einer Textzeile, welche den *Paketnamen* beschreibt.

Weiterhin existieren zusätzliche Knoten, welche keine Softwarebausteine repräsentieren (siehe Abbildung 2.3). Hierzu zählen unter anderem die in *mehrgliedrigen Assoziationen* auftretenden Rauten. Diese verbinden über Kanten mindestens drei weitere Knoten und enthalten, im Gegensatz zu den bisher präsentierten Elementen, keinerlei Text. Des Weiteren existiert die *Notiz*, welche gänzlich der Diagrammbeschreibung dient. Diese wird ebenso durch ein Rechteck symbolisiert, wobei in der Regel die obere rechte Ecke gebogen ist. Die Notiz enthält zudem zwangsläufig einen Textblock, welcher das Ziel der von ihr ausgehenden Kante beschreibt. Diese wird im Folgenden als *Notizverbindung* referenziert.

Kanten Neben den genannten Knoten bestehen UML-Klassendiagramme aus einer Vielzahl von Kantentypen. Diese sind dabei nicht immer zwangsläufig gerichtet wie in Diagrammdatensätzen verwandter Arbeiten [14, 15, 19]. So ist das Auftreten einer Pfeilspitze bei den Kantentypen *Assoziation*, *Aggregation* und *Komposition* nur optional. Bei Assoziationen ohne Pfeilspitze, die lediglich aus einer durchgezogenen Linie bestehen, führt dies zwangsläufig zu einer bidirektionalen Verbindung. Aggregationen und Kompositionen bestehen hingegen zusätzlich aus einer sich an der Kantenquelle befindenden Raute. Dadurch

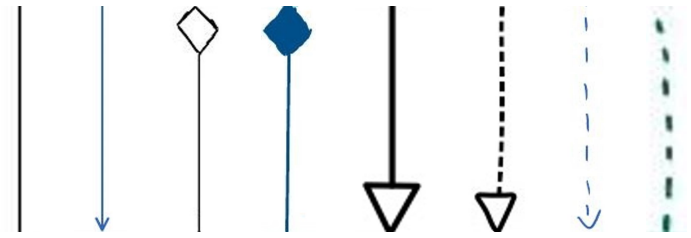


(a) **(b)** **(c)**

Abbildung 2.3.: **(a)** Paket, **(b)** Notiz und **(c)** Raute einer dreigliedrigen Assoziation aus jeweils verschiedenen Skizzen des in dieser Arbeit präsentierten Datensatzes

sind sie auch im Falle der Nichtexistenz einer Pfeilspitze gerichtet, da die Richtung der Verbindung klar ersichtlich ist. Über die genannte Raute erfolgt zusätzlich die Separation zwischen Aggregation und Komposition. Diese ist bei Letzteren farblich ausgefüllt, bei Erstgenannten hingegen nicht. Alle drei genannten Kantentypen können zudem über einen *Qualifizierer* verfügen, welcher zusätzlich an der Kantenquelle liegt. Dieser wird durch ein Rechteck visualisiert und enthält einen Textblock.

Weitere, in UML-Klassendiagrammen existierende Kantentypen, sind *Vererbungen*, *Realisierungen* und *Abhängigkeiten*. Diese besitzen in jedem Fall eine Pfeilspitze und sind folglich immer gerichtet. Die Unterscheidung dieser Elemente kann sowohl in der Pfeilspitze als auch in der Linienform erfolgen, welche gepunktet oder durchgezogen sein kann (siehe Abbildung 2.4).



(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)** **(h)**

Abbildung 2.4.: **(a)** Ungerichtete und **(b)** gerichtete Assoziation, **(c)** Aggregation, **(d)** Komposition, **(e)** Vererbung, **(f)** Realisierung, **(g)** Abhängigkeit und **(h)** Notizverbindung aus jeweils verschiedenen Skizzen des in dieser Arbeit präsentierten Datensatzes. Die Kanten verlaufen von oben (Quelle) nach unten (Ziel).

Analog zu den bereits präsentierten Knoten können Kanten ebenfalls Text besitzen. Assoziationen, Aggregationen und Kompositionen können so an beiden ihrer Enden über eine *Multiplizität* verfügen. Diese gibt an, wie oft die verbundenen Knoten über diese

Beziehung in Relation zueinander stehen dürfen und repräsentiert daher immer eine Zahlenmenge. Text dieser Art lässt sich sowohl über seine Position als auch über einen regulären Ausdruck klassifizieren. Bei weiteren Textarten, wie beispielsweise der *Rolle* oder einem *Assoziationsnamen*, ist dies nicht immer möglich. Hier ist oft der Gesamtkontext und die semantische Bedeutung für die Klassifikation ausschlaggebend, auf welche eine maschinelle Erkennung keinen Zugriff hat. Folglich wird in dieser Arbeit jede weitere Art von Kantentext zu einer Textkategorie mit dem Namen *Kantenbeschriftung* zusammengefasst. Jeder Kantentyp kann dabei eine oder mehrere Kantenbeschriftungen besitzen.

Die UML verfügt zudem über einige syntaktische Regeln, welche den erlaubten Einsatz der bereits beschriebenen Kanten definieren. Eine Realisierung hat so beispielsweise immer eine Schnittstelle als Ziel, wobei ihre Quelle eine Klasse oder abstrakte Klasse sein muss. Da sich Skizzen aber in der Regel eher lose an die exakte Spezifikation der UML halten [2], sind diese syntaktischen Regeln für den Aufbau des Datensatzes von geringerer Relevanz. Eine genauere Betrachtung und Definition dieser ist daher für den Zweck der Arbeit nicht nötig.

2.1.2. Diagrammskizzen: Probleme und Herausforderungen

Handgezeichnete Skizzen bringen im Vergleich zu von CASE-Tools erstellten Diagrammen einige Besonderheiten mit sich, die im Folgenden illustriert werden. Diese können eine automatisierte, maschinelle Erkennung von Diagrammelementen und deren Struktur erschweren.

Werden Diagramme digital erzeugt, so variiert die Darstellung der enthaltenen Elemente ausschließlich zwischen den zugehörigen CASE-Tools. Auf diese Weise generierte Klassendiagramme sind standardisiert, das heißt jeder Elementtyp weist innerhalb eines gleichen CASE-Tools die exakt gleiche geometrische Form auf (siehe Abbildung 2.5).

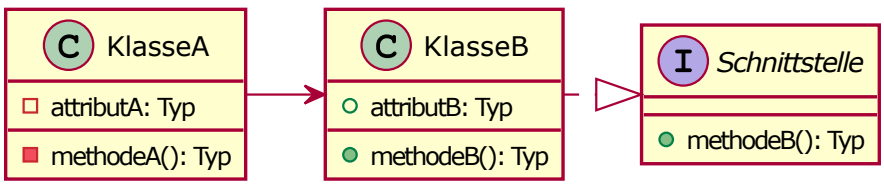


Abbildung 2.5.: Beispiel eines von einem CASE-Tool erzeugten UML-Klassendiagramms. Alle Knoten und Kanten werden akkurat und präzise dargestellt, Elemente gleicher Art besitzen die exakt gleiche geometrische Form.

Es ist dabei nicht von Relevanz, welche Person dieses Diagramm erstellt hat. Bei handgezeichneten Diagrammen ist aber genau dies ein wichtiger Faktor, der maßgeblich für die Elementgestalt verantwortlich ist. Diese Variation äußert sich nicht nur an der unterschiedlichen Darstellung einzelner Elemente. Faktoren wie der *Diagrammhintergrund*, das *Zeichengerät* oder auch die *Aufnahmequalität* beeinflussen ebenso das Erscheinungsbild des dargestellten Diagramms. Ersteres äußert sich bei handgezeichneten Diagrammen vor allem am zugrundeliegenden Papiertyp. Kariertes, liniertes oder punktiertes Papier bringen so zusätzliche Linien in die zu erkennende Skizze. Diese müssen von tatsächlich

gezeichneten Linien separiert werden und erschweren somit das Erkennen von relevanten Elementformen.

Eine in Skizzen ebenfalls existierende Problematik sind häufige Ungenauigkeiten der Zeichner (siehe Abbildung 2.6). So werden Kanten oder Knoten nicht immer vollständig zu Ende gezeichnet. Auch das Existieren von durchgestrichenen Elementen oder Text ist eine in Diagrammskizzen auftretende Eigenschaft. Diese können das Erkennen relevanter Elemente behindern oder fälschlicherweise zum Erkennen nicht existierender Elemente führen. In von CASE-Tools erzeugten Diagrammen treten diese Probleme in keinster Weise auf. Diese sind durch eine hohe Präzision ausgezeichnet, das heißt alle Knoten treten immer in ihrer vollen Form auf und alle Kanten starten und enden exakt an den zugehörigen Knotenpunkten. Des Weiteren können getätigte Fehler bei der Diagrammerstellung vollständig entfernt werden, so dass im fertigen Diagramm keinerlei Rückschlüsse auf diese existieren. All die genannten Punkte betreffen dabei nicht nur die Domäne der UML-Klassendiagramme, sondern sind generell im Bereich der Diagrammskizzen relevant. In der Arbeit von Schäfer u. a. [15] werden die genannten Faktoren bei der Erkennung von *Business Process Model and Notation* (BPMN)-Skizzen ebenso illustriert, und es wird gezeigt, dass diese eine wichtige Rolle bei der Erkennung handgezeichneter Diagramme spielen.

(a) **(b)**

Abbildung 2.6.: **(a)** Ungenau gezeichnete Kanten, **(b)** nicht fertig gezeichnete und durchgestrichene Knoten aus jeweils verschiedenen Skizzen des in dieser Arbeit präsentierten Datensatzes

Aus den illustrierten Herausforderungen lässt sich folgern, dass eine maschinelle Erkennung handgefertigter Klassendiagramme durch zusätzliche Parameter erschwert wird. Des Weiteren zeigt es ebenso wie Abbildung 2.1 und Abbildung 2.2, dass sich Diagrammskizzen in ihrer Form von CASE-Tool generierten Diagrammen unterscheiden können. Das Trainieren eines Datensatzes CASE-Tool generierter Klassendiagramme würde folglich nicht das Ausgangsproblem aus Kapitel 1 lösen, da diese die genannten Faktoren nicht berücksichtigen. Dies zeigt beispielsweise auch die Arbeit von Huber und Hagel [20], in der das Training von hauptsächlich digital erzeugten Diagrammen die Erkennung handgezeichneter Klassendiagramme nicht zufriedenstellend lösen konnte. Der in dieser Arbeit präsentierte Datensatz fokussiert sich daher ausschließlich auf UML-Klassendiagrammskizzen.

2.2. Erkennen von handgezeichneten Diagrammen

In den letzten Jahren haben sich verschiedene Möglichkeiten zur automatisierten Erkennung von handgezeichneten Diagrammen entwickelt. Hierbei lässt sich vor allem zwischen zwei Ansätzen unterscheiden, der *Online-* und der *Offline-Erkennung*. Im Folgenden werden die Gemeinsamkeiten und Unterschiede dieser Lösungsansätze illustriert. Des Weiteren wird dargestellt, in welche dieser Richtungen sich der Anwendungsbereich des in dieser Arbeit präsentierten Datensatzes einordnen lässt.

Bei der Online-Erkennung von Diagrammen wird eine Skizze in der Regel auf einem digitalen Eingabegerät gezeichnet. Dadurch ist es möglich, die vom Endnutzer gezeichneten Striche mit temporalen Informationen aufzunehmen und zu speichern. Mit diesen ist es auch nach Diagrammentstehung weiterhin nachvollziehbar, in welcher Reihenfolge Punkte und Striche gezeichnet wurden. Diese zusätzlichen Informationen können bei der Erkennung von Elementsymbolen berücksichtigt werden und unterstützen diese maßgeblich. Einen solchen oder ähnlichen Ansatz haben bereits zahlreiche Arbeiten im Bereich der UML-Klassendiagramme [21, 22, 23] verfolgt. Auch in anderen Diagrammdomänen, wie die der Flussdiagramme, haben Verfahren der Online-Erkennung von Skizzen bereits Anwendung gefunden [24].

Innerhalb der Offline-Erkennung existieren wiederum verschiedene Lösungsansätze. Diese haben den Vorteil, dass im Gegensatz zur Online-Erkennung nur ein Bild eines gezeichneten Diagramms vorliegen muss. Das Vorhandensein zusätzlicher Informationen der vom Zeichner getätigten Striche ist somit nicht essenziell. Aus diesem Grund können auch Kameraaufnahmen oder eingescannte Diagramme problemlos verarbeitet werden. Die Skizze kann also sowohl auf einem digitalen als auch auf einem nicht-digitalen Medium entstanden sein. Die im Folgenden vorgestellten Ansätze sind daher bezüglich ihrer Eingabe höchst flexibel, so dass auch die Verarbeitung alter Diagrammskizzen ohne Hintergrundinformationen möglich ist.

Eine Möglichkeit der Offline-Erkennung von Diagrammelementen ist die Rekonstruktion von im Bild enthaltenen Strichen wie in der Arbeit von Bresler, Průša und Hlaváč [19]. Nach dieser ist der Einsatz vorhandener Online-Erkennungsmethoden möglich, solange diese lediglich über räumliche und nicht zwingend über temporale Strichinformationen verfügen müssen. Eine zweite Möglichkeit ist das Nutzen maschineller Lernverfahren zur Objekterkennung. Verschiedene Deep-Learning Objektdetektoren wurden in diesem Bereich in den letzten Jahren vermehrt für die Erkennung von Diagrammelementen eingesetzt [13, 14, 15, 16, 20]. Verfahren dieser Art werden dabei zunächst auf einer Vielzahl von Daten trainiert und anschließend auf die Erkennung vom Trainingssatz disjunkter Skizzen geprüft. Diese Daten müssen hierbei stets annotiert sein, das heißt alle im Bild enthaltenen Elemente sind durch eine Klassifizierung des Elementtyps und durch eine *Bounding-Box* gekennzeichnet. Bounding-Boxen umgeben relevante Objekte minimal und definieren somit im Diagrammkontext Position und Größe der enthaltenen Elementarten. Julca-Aguilar und Hirata [13] haben mit einem solchen Ansatz *Faster R-CNN* [25], ein sogenanntes *Region Based Convolutional Neural Network* (R-CNN), auf die Erkennung von handgezeichneten Symbolen in Flussdiagrammen und mathematischen Ausdrücken validiert. Die hier entstandenen Ergebnisse zeigen, dass das Trainieren derartiger Objektdetektoren die Offline-Erkennung handgefertigter Symbole gewährleisten kann. Für eine Erkennung der

Gesamtstruktur einer Diagrammskizze ist dies jedoch ungenügend. Essenziell sind hier sogenannte *Kantenschlüsselpunkte*. Diese definieren im Falle einer unidirektionalen Kante genau die Punkte, an denen die Kante an ihrer Quelle und an ihrem Ziel aus- und eingeht. Der Objektdetektor Arrow R-CNN [14] erweitert aus diesem Grund Faster R-CNN um eine Vorhersage der genannten Schlüsselpunkte. Dies ermöglicht die Anwendung im Bereich ganzer Diagrammskizzen. Ergebnisse in Bereichen von handgezeichneten Flussdiagrammen [14], endlichen Automaten [14] und BPMN-Diagrammen [15] demonstrieren dabei signifikante Verbesserungen des damaligen State-of-the-Art der Skizzenerkennung. Dies äußert sich unter anderem an einer erheblich besseren Erkennungsrate als bestehende Online-Erkennungsmethoden auf Basis gleicher Daten. Des Weiteren werden auch die Offline-Erkennungsergebnisse der Arbeit von Bresler, Průša und Hlaváč [19] übertroffen, welche auf die Rekonstruktion von Strichen und den anschließenden Einsatz bestehender Online-Erkennungsmethoden aufbaut. Arrow R-CNN habe jedoch bei der Erkennung gerader Pfeile mit engen Bounding-Boxen und der von Pfeilen, deren Bounding-Box stark mit der von anderen Kanten überlappt, noch Schwierigkeiten. Aus diesem Grund stellen Schäfer und Stuckenschmidt [16] in einer weiteren Arbeit das *DiagramNet* vor. Dieses löst das genannte Problem durch ein alternatives Erkennungsverfahren von Pfeilen. Die erzielten Ergebnisse übertreffen die mit Arrow R-CNN erreichten Erkennungsraten auf Basis der gleichen Datensätze speziell bei Kanten deutlich.

Ein Einsatz ähnlicher Verfahren im Bereich der UML-Klassendiagramme ist aufgrund fehlender, annotierter Datensätze bis jetzt nur im geringem Maße untersucht worden. In der Arbeit von Huber und Hagel [20] wird das Objekterkennungssystem *YOLO* (You Only Look Once) [26] auf die Erkennung von handgezeichneten UML-Klassendiagrammen geprüft. Dieses wurde allerdings hauptsächlich auf von dem CASE-Tool *Enterprise Architect* generierten Klassendiagrammen trainiert. Eine Erkennung sei daher nur in sehr ordentlich gezeichneten Skizzen darstellbar, wobei Diagrammskizzen in der Praxis allerdings meist eher unpräzise sind (siehe Unterabschnitt 2.1.2). Das vollständige Trainieren eines Datensatzes von händisch gezeichneten UML-Klassendiagrammen könnte hier zu verbesserten Ergebnissen führen. Ein erfolgreicher Einsatz eines maschinelles Lernverfahrens wie bei DiagramNet ist daher unter Verfügbarkeit eines Datensatzes von UML-Klassendiagrammskizzen ebenfalls denkbar. Aus diesem Grund soll der in dieser Arbeit präsentierte Datensatz initial auf einem angepassten DiagramNet-Ansatz eingesetzt werden (siehe Abschnitt 6.2).

2.3. COCO-Datenformat

Der Microsoft *Common Objects in Context* (COCO)-Datensatz [17] ist ein im Jahre 2015 veröffentlichter Datensatz, welcher das Ziel hatte, die automatisierte Erkennung verschiedenster Objekte voranzuschreiten. Bestehend aus 328.000 Bildern werden unterschiedliche Situationen dargestellt, in welchen 91 verschiedene Objektkategorien enthalten sind. Diese umfassen dabei beispielsweise *Personen*, verschiedenste *Tiere* oder *Gegenstände*. Alle in den Bildern auftauchenden Objekte wurden manuell mit einer passenden Kategorie und Bounding-Box annotiert. Die auf diese Weise entstandenen Annotationen werden gemein-

sam mit Bildinformationen und den definierten Objektkategorien in einem passenden JSON-Format [18], im Folgenden als *COCO-Datenformat* referenziert, gespeichert.

Dieses kann dabei flexibel durch weitere Kategorien erweitert werden, so dass ein Einsatz dieses Formats in weiteren Domänen der Objekterkennung möglich ist. Auch im Bereich der Skizzenerkennung ist die Anwendung des COCO-Datenformats möglich. Ein Beispiel hierfür ist der *hdBPMN*-Datensatz [27] von der *Data and Web Sience Group* (dwslab) der Universität Mannheim aus dem Jahre 2020, welcher aus händisch gezeichneten BPMN-Diagrammen und ihren zugehörigen Annotationen besteht. Mit Hilfe der im COCO-Datenformat gesicherten Daten wurden in den Arbeiten von Schäfer u. a. [15, 16] gezeigt, dass maschinelle Lernverfahren nach Trainieren einer Teilmenge des Datensatzes die automatisierte Erkennung von BPMN-Skizzen gewährleisten können. Der in dieser Arbeit präsentierte Datensatz soll analog dazu die Skizzenerkennung von UML-Klassendiagrammen unterstützen. Aus diesem Grund werden die Skizzen nach ihrer Erhebung und Annotation ebenfalls in ein erweitertes COCO-Format konvertiert. Im Folgenden wird das hiermit verbundene Schema auf den UML-Klassendiagrammkontext übertragen und dargestellt. Dies wird dabei beispielhaft mit der UML-Klassendiagrammskizze aus Abbildung 1.1 demonstriert.

Das COCO-Datenformat besteht aus drei größeren Blöcken, die jeweils der Beschreibung der im Datensatz vorkommenden *Bilder*, *Annotationen* und *Kategorien* dienen. Der vorderste Block definiert dabei alle Bilddateien. Zu jedem Bild wird dementsprechend sowohl der Dateiname als auch die Bildgröße mit einem eindeutigen Identifikator gespeichert. Abbildung 2.7 illustriert dies am Beispiel der Diagrammskizze aus Abbildung 1.1.

```
"images": [
    {
        "file_name": "exampleDiagram.jpg",
        "height": 410,
        "width": 1679,
        "id": 0
    }
]
```

Abbildung 2.7.: Speicherung des Bildes aus Abbildung 1.1 im COCO-Datenformat

Die Bild-ID wird benötigt, um Annotationen einem Bild eindeutig zuweisen zu können. Diese werden im darauffolgenden Block der JSON-Datei dargestellt. Jede der dort beschriebenen Annotationen besitzt ebenfalls eine eindeutige ID und wird über *„category“* und *„category_id“* exakt einer Kategorie zugeordnet. Das Feld *„bbox“* beschreibt zusätzlich die Position und Größe der zugehörigen Bounding-Box. Dieses enthält dabei gemäß folgender Reihenfolge die *X-Koordinate*, *Y-Koordinate*, *Breite* und *Höhe* der Bounding-Box in Pixel. Der weitere Aufbau der Annotationen unterscheidet sich im Kontext von UML-Klassendiagrammen zwischen Knoten, Kanten und Text. So ist das Feld *„keypoints“* in der Diagrammdomäne ausschließlich für Kanten relevant. Für Knoten und Text sind die enthaltenen Werte folglich immer 0 und können vernachlässigt werden. Abbildung 2.8 zeigt exemplarisch die Annotation einer Klasse aus Abbildung 1.1.

```
{
    "id": 0,
    "image_id": 0,
    "category": "Class",
    "category_id": 0,
    "area": 156603.366,
    "bbox": [11.753, 16.79, 416.392, 376.096],
    "keypoints": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
}
```

Abbildung 2.8.: Speicherung einer in Abbildung 1.1 dargestellten Klasse im COCO-Datenformat

Bei annotierten Kanten ist das Feld „keypoints“ hingegen von enormer Wichtigkeit, da es die zugehörigen Kantenschlüsselpunkte repräsentiert. Des Weiteren werden mit *„arrow_prev“* und *„arrow_next“* die Quelle und das Ziel der beschriebenen Kante über die jeweiligen Knotenidentifikatoren referenziert. Die zusätzlich auftretenden *„waypoints“* definieren darüber hinaus die exakten Wegpunkte der Kante. Alle der bisher eingeführten Werte treten analog ebenfalls im hdBPMN-Datensatz auf. Die Felder *„has_arrowhead“* und *„directed“* existieren hingegen nur im Kontext der UML-Klassendiagramme. Diese beschreiben den Status der Kantenrichtung, da Verbindungen hier nicht zwangsläufig gerichtet sind wie in der BPMN [28]. Abbildung 2.9 präsentiert beispielhaft die Annotation einer unidirektionalen Assoziation aus Abbildung 1.1.

```
{
    "id": 2,
    "image_id": 0,
    "category": "AssociationUnidirectional",
    ...,
    "keypoints": [428.145, 181.332, 2.0, 693.427, 174.616, 2.0],
    "arrow_prev": 0,
    "arrow_next": 1,
    "waypoints": [[428.145, 181.332], [693.427, 174.616]],
    "has_arrowhead": "true",
    "directed": "true"
}
```

Abbildung 2.9.: Speicherung einer in Abbildung 1.1 dargestellten, unidirektionalen Assoziation im COCO-Datenformat

Die Annotation von Textblöcken ist der von Knoten ähnlich. Der Unterschied äußert sich an den zusätzlichen Feldern *„label_type“*, *„belongs_to“* und *„name“*. Die zwei Letztgenannten sind ebenfalls Teil des hdBPMN-Datensatzes. Die *„belongs_to“* Relation speichert dabei den Identifikator des vom Text beschriebenen Elements und *„name“* den zugehörigen Textinhalt. Der im UML-Kontext zusätzlich existierende *„label_type“* gibt hingegen an,

welche der in Unterabschnitt 2.1.1 definierten Textkategorien vorliegt. Abbildung 2.10 illustriert exemplarisch den Aufbau einer Textannotation anhand eines Klassennamens aus Abbildung 1.1.

```
{
    "id": 3,
    "image_id": 0,
    "category": "Label",
    ...,
    "belongs_to": 0,
    "label_type": "name",
    "name": "KlasseA"
}
```

Abbildung 2.10.: Speicherung von Text in Form eines Klassennamens aus Abbildung 1.1 im COCO-Datenformat

Der letzte im COCO-Format bestehende Block beschreibt alle in der Domäne existierenden Objektkategorien. Bei UML-Klassendiagrammen sind dies genau die in Unterabschnitt 2.1.1 beschrieben Knoten- und Kantentypen. Zusätzlich werden Textblöcke durch die Kategorie *„Label“* repräsentiert. Jede Kategorie besitzt zudem eine Superklasse. Knoten erweitern so die Kategorie *„Shape“* und Kanten die Kategorie *„Edge“*. Text erweitert im Grunde keine der bestehenden Kategorien und besitzt somit als Superklasse wiederum *„Label“*. Das bei Kantenkategorien zusätzlich auftretende Feld *„keypoints“* spezifiziert zudem die Bedeutung ihrer Schlüsselpunkte. Abbildung 2.11 zeigt die Definition einer in Abbildung 1.1 auftretenden Kantenkategorie, um den Aufbau dieser weiter zu illustrieren.

```
"categories": [
    {
        "supercategory": "Edge",
        "id": 18,
        "name": "AssociationUnidirectional",
        "longname": "Unidirectional Association",
        "keypoints": ["head", "tail"]
    }
]
```

Abbildung 2.11.: Definition einer Kantenkategorie im COCO-Datenformat. Die Schlüsselpunkte werden über *„head“* und *„tail“* als Ziel- und Quellpunkt der Kante definiert.

3. Verwandte Arbeiten

Im Folgenden werden verwandte Arbeiten vorgestellt, welche Datensätze aus den Bereichen der UML und der Skizzenerkennung präsentieren. Die in Abschnitt 3.1 thematisierten Arbeiten beschäftigen sich mit existierenden Datensätzen von UML-Klassendiagrammen und deren Anwendungszweck. Diese müssen dabei nicht ausschließlich der automatisierten Diagrammerkennung dienen, sondern können ebenso andere Ziele verfolgen. In Abschnitt 3.2 werden Arbeiten präsentiert, welche sich mit der Offline-Erkennung von händisch gezeichneten UML-Klassendiagrammen auseinandersetzen. Im Zuge dessen werden die für die Evaluierung dieser Verfahren verwendeten Daten dargestellt. Zuletzt werden in Abschnitt 3.3 weitere verwandte Arbeiten vorgestellt, welche sich explizit mit der Erkennung von handgezeichneten Diagrammen mittels maschineller Lernverfahren beschäftigen. Der Fokus der Betrachtung liegt dabei vor allem auf den zu diesen Zwecken eingesetzten Datensätzen. Diese enthalten allesamt Diagramme, welche nicht aus dem Bereich der UML stammen.

3.1. UML-Datensätze

Trotz der weiten Verbreitung und dem Gelten als Industriestandard für das Modellieren von Software [5] sind öffentliche Datensätze von UML-Klassendiagrammen relativ rar. Es existieren dennoch nennenswerte Ansammlungen von UML-Klassendiagrammdaten, welche nicht zwangsläufig für den Zweck der automatisierten Diagrammerkennung entstanden sind. Im Folgenden werden zwei dieser Datensätze präsentiert und von dem in dieser Arbeit vorgestellten Datensatz handgezeichneter UML-Klassendiagramme abgegrenzt.

Das *Img2UML-Repository* [29] von Karasneh und Chaudron aus dem Jahre 2013 ist eine Datenbank bestehend aus 1.000 UML-Klassendiagrammen. Das Ziel der Autoren ist es, durch die offene Verfügbarkeit dieser Diagramme, sowohl die Analyse als auch das Studieren von UML-Klassendiagrammen für Forschung, Industrie und Lehre zu erleichtern. Die in der Datenbank enthaltenen Diagramme wurden dabei jeweils aus dem Internet als Bilddatei erhoben. Wie bereits in Kapitel 1 illustriert, wird eine digitale Weiterverarbeitung eines ausschließlich auf einem Bild gespeicherten Diagramms erschwert. Karasneh und Chaudron [29] bekräftigen dies und stellen daher zusätzlich eine zum Diagrammbild zugehörige XMI-Datei bereit. Diese kann dabei als Eingabe für das CASE-Tool Star-UML [9] dienen, mit welchem das Diagramm digital verarbeitet werden kann. Für die automatisierte Erstellung dieser XMI-Datei stellen Karasneh und Chaudron in einer weiteren Arbeit das Erkennungssystem *Img2UML* [30] vor. Das hier entstandene Verfahren ist in der Lage, aus einem eingegebenen Bild eines UML-Klassendiagramms eine mit Star-UML kompatible XMI-Datei zu generieren. Der Einsatz beschränkt sich jedoch auf von CASE-Tools erzeugte Klassendiagrammbilder. Die Erkennung handgezeichneter Diagramme ist somit nicht

ohne Weiteres möglich. Aus gleichem Grund enthält das Img2UML-Repository keinerlei Klassendiagrammskizzen. Die Datenbank eignet sich daher nicht für den Einsatz eines maschinelles Lernverfahren zur Erkennung von handgezeichneten UML-Klassendiagrammen, da sich von CASE-Tools erzeugte Diagramme stark von Skizzen unterscheiden (siehe Unterabschnitt 2.1.2). Des Weiteren liegen die gespeicherten Daten noch nicht in einem passenden Format für das maschinelle Lernen dieser vor. Eine Konvertierung der Bild- und XMI-Dateien in ein kompatibles Format, wie beispielweise das COCO-Datenformat (siehe Abschnitt 2.3), wäre daher selbst bei Erhalt von händisch gezeichneten Diagrammen weiter notwendig.

In der Arbeit von Robles u. a. [31] wird der *Lindholmen Datensatz* vorgestellt. Dieser beschränkt sich nicht ausschließlich auf Klassendiagramme und besitzt Verweise auf über 93.000 UML-Diagramme aller Art, wovon 57.822 Bilddateien sind. Ähnlich wie beim Img2UML-Repository stammen die zugrundeliegenden Daten aus dem Internet. Der Datensatz speichert so in Form einer Datenbank Direktverweise auf *GitHub*-Projekte, welche UML-Dateien beinhalten. Dies soll das Studieren ganzer Softwareprojekte ermöglichen, in welchen das Modellieren von Softwaresystemen mittels UML ein elementarer Bestandteil ist. Eine Datei ist dabei eine UML-Datei, wenn sie ein Standard-UML-Format oder einen UML-typischen Namen besitzt. Auf diese Art und Weise werden gleichermaßen CASE-Tool generierte Diagramme und handgezeichnete UML-Diagramme identifiziert. Folglich enthält der Lindholmen Datensatz im Gegensatz zum Img2Uml-Repository auch Klassendiagrammskizzen. Aus diesem Grund ist dieser für den Anwendungsbereich der automatisierten Erkennung von handgezeichneten UML-Klassendiagrammen durchaus relevant. Die im Datensatz enthaltenen Skizzen sind jedoch nicht mit Bounding-Boxen annotiert, das heißt es existieren keinerlei Informationen über Position und Größe aller vorhandenen Diagrammelemente. Des Weiteren besitzt die Datenbanktabelle keinen Eintrag über den Ursprung der Diagrammerzeugung. Für die Identifizierung von Skizzen ist daher eine weitere, manuelle Betrachtung aller im Datensatz enthaltenen Bilddateien notwendig. Der Lindholmen-Datensatz kann aus diesen Gründen zwar nicht selbstständig ohne Aufwand für das maschinelle Lernen eingesetzt werden, kann aber potenziell Quelle für weitere Datensätze dieses Anwendungszwecks sein. In Abschnitt 4.3 wird daher untersucht, inwieweit sich dieser als Teilquelle für den in dieser Arbeit präsentierten Datensatz handgezeichneter UML-Klassendiagramme eignet. Ebenso wird dabei verdeutlicht, dass dies nur mit einem hohen, manuellen Aufwand möglich ist.

Sowohl das Img2Uml-Repository als auch der Lindholmen-Datensatz können nicht direkt für das automatisierte Erkennen von UML-Klassendiagrammskizzen verwendet werden. Nach aktuellem Kenntnisstand sind daher derzeit keine annotierten UML-Datensätze veröffentlicht, welche sich für die direkte Eingabe eines maschinellen Lernverfahrens eignen.

3.2. Offline-Erkennung von UML-Klassendiagrammen

Es existieren trotz des in Abschnitt 3.1 beschriebenen Mangels an öffentlichen Datensätzen einige Arbeiten, welche sich mit der Offline-Erkennung von UML-Klassendiagrammen auseinandersetzen. Deufemia und Risi [23] stellen so in ihrer Arbeit einen Ansatz für

die Erkennung von handgezeichneten Diagrammen vor, welcher sich auf mehrere Domänen übertragen lässt. Das System nutzt dafür im Vorfeld definierte Grammatiken, um domänenspezifische Symbolformen und Diagrammregeln zu modellieren. Dadurch ist es sowohl für die Online- als auch für die Offline-Erkennung nutzbar. Im Falle der Online-Erkennung erkennt das Verfahren in einem ersten, domänenunabhängigen Schritt anhand der gezeichneten Linien primitive Formen wie Ellipsen oder Bögen. Bei der Offline-Erkennung wird anstatt der Linien die zugrundeliegende Bitmap des Eingabebildes analog verwendet. In einem nächsten Schritt werden die erkannten Formen über die vorher definierten Grammatiken der einzelnen Symbolformen klassifiziert. Anschließend werden die ebenfalls im Vorfeld definierten Diagrammregeln zur Verfeinerung dieser Klassifizierungen angewandt. Das Verfahren wird in der Arbeit beispielhaft auf die Domäne der UML-Klassendiagramme validiert und kann im Bereich der Online-Erkennung vielversprechende Ergebnisse vorweisen. Die für den Anwendungszweck dieser Bachelorarbeit eher interessante Offline-Erkennung wird hingegen nicht evaluiert. Deufemia und Risi begründen dies nicht zwangsläufig mit dem Mangel an passenden Datensätzen von UML-Klassendiagrammskizzen. Ein vollständig annotierter Datensatz könnte jedoch eine Validierung in zukünftigen Arbeiten beschleunigen, da man sich so eine eigenständige Erhebung und Annotation sparen könnte.

In der bereits in Unterabschnitt 2.1.2 angesprochenen Arbeit von Huber und Hagel [20] wird ein Ansatz präsentiert, dessen Ziel es ist, die syntaktische Korrektheit von UML-Klassendiagrammen mittels maschinellem Lernen automatisiert zu überprüfen. Um einzelne UML-Elemente zu erkennen und zu lokalisieren, wird das Objekterkennungssystem *You Only Look Once (YOLO)* [26] eingesetzt, welches zuvor mit 1.000 Klassendiagrammbildern trainiert wurde. Darüber hinaus werden weitere Deep-Learning-Verfahren zur Texterkennung und der Bestimmung von Kantenrichtungen angewandt. Das Lernverfahren wurde bislang ausschließlich mit vom CASE-Tool Enterprise Architect [8] erzeugten Diagrammen trainiert. Die hierfür genutzten Trainingsdaten wurden dabei selbstständig erzeugt und nicht veröffentlicht. Nach dem Training registrierte, exakte Erkennungsraten von UML-Klassendiagrammen sind zum jetzigen Zeitpunkt ebenfalls nicht öffentlich. Das Erkennen einzelner UML-Knoten und Kanten in digital erzeugten Diagrammen sei derzeit laut den Autoren nur bis zu einem gewissen Grad möglich. In einigen Fällen war YOLO demzufolge nicht in der Lage, alle sich im Bild befindenden UML-Elemente ausfindig zu machen. Das System habe außerdem Schwierigkeiten, Quell- und Zielknoten von Kanten bei Diagrammen zuzuordnen, in welchen viele Elemente eng beieinander liegen. Eine Erkennung handgezeichneter UML-Elemente sei gar nur bei äußert ordentlichen Zeichnungen darstellbar gewesen. Huber und Hagel sind dabei der Meinung, dass durch das vermehrte Training handgezeichneter Diagramme die Erkennung von Skizzen noch Verbesserungspotential besitze. Dies verdeutlicht wiederum den Bedarf an passenden Datensätzen, welche sich ausschließlich auf Klassendiagrammskizzen fokussieren.

3.3. Datensätze für die Diagrammerkennung mittels maschineller Lernverfahren

In anderen Diagrammdomänen existieren zum aktuellen Zeitpunkt bereits Datensätze, welche die automatische Erkennung von Skizzen mittels maschineller Lernverfahren unterstützen können. Diese Datensätze wurden dabei in mehreren Arbeiten eingesetzt, in welchen nennenswerte Erkennungsergebnisse erreicht werden konnten.

In der Arbeit von Julca-Aguilar und Hirata [13] wird Faster R-CNN als Methode zur Offline-Erkennung von Symbolen in handgezeichneten Flussdiagrammen und mathematischen Ausdrücken evaluiert. Für den Anwendungsbereich dieser Bachelorarbeit ist vor allem das Erkennen von Flussdiagrammskizzen von Relevanz. Die dafür zugrundeliegende Daten stammen aus dem *Online-Datensatz* von Awal u. a., welcher über 419 Flussdiagramme verfügt. Online-Datensätze enthalten dabei lediglich Informationen über die gezeichnete Strichreihenfolge der enthaltenen Symbole. Für eine Erkennung mittels maschinellem Lernen sind jedoch *Offline-Datensätze* notwendig, welche mit Bounding-Boxen annotierte Bilder von Skizzen enthalten. Aus diesem Grund haben Julca-Aguilar und Hirata über die gegebenen Strichinformationen in einem Zwischenschritt Offline-Daten rekonstruiert. Eine Teilmenge dieser hat anschließend Faster R-CNN trainiert, während die restlichen Offline-Daten der späteren Validierung dienten. Die hierbei entstandenen Erkennungsergebnisse zeigen, dass Faster R-CNN in der Lage ist, effektiv einzelne Diagrammsymbole in einem Flussdiagramm zu erkennen. Die Erkennung der Gesamtstruktur eines Diagramms ist mit diesem Verfahren, wie bereits in Abschnitt 2.2 dargestellt, noch nicht möglich. Dennoch zeigt es, dass maschinelle Lernverfahren die automatisierte Erkennung von handgezeichneten Elementen ermöglichen können. Außerdem belegt es, dass ein Einsatz von bestehenden Online-Daten für Offline-Verfahren möglich sein kann. Das Erstellen eines natürlichen Offline-Datensatzes ist für das Ziel der automatisierten Skizzenerkennung jedoch naheliegender, da man sich die Zwischenschritte der Konvertierung erspart. Des Weiteren ignorieren ursprüngliche Online-Daten Herausforderungen, die in der natürlichen Entstehung von Skizzen auftreten können. Dazu zählen unter anderem die in Unterabschnitt 2.1.2 dargestellten Variationen bezüglich Diagrammhintergrund, Zeichengerät und Aufnahmequalität. Speziell beim hier verwendeten Online-Datensatz von Awal u. a. ist zudem fraglich, ob die enthaltenen Online-Skizzen praxisnah sind und somit eine externe Validität besteht. Die im Datensatz enthaltenen 419 Skizzen wurden unter Vorlage von sieben Flussdiagrammen von 36 verschiedenen Zeichnern abgezeichnet. Eine Variation bezüglich Diagrammstruktur innerhalb einer Vorlage wird dabei nur durch das Umstrukturieren von Diagrammelementen gegeben. Zusätzlich sind diese sehr ordentlich und exakt, was gegen den natürlichen Aufbau einer Skizze spricht (siehe Unterabschnitt 2.1.2).

Der Datensatz von Awal u. a. [32] findet ebenfalls in der Arbeit von Schäfer, Keuper und Stuckenschmidt Anwendung, in welcher das Objekterkennungssystem Arrow R-CNN initial evaluiert wird. Dieser wird dabei ebenso wie zwei weitere Online-Datensätze von Flussdiagrammskizzen und einem Online-Datensatz von handgezeichneten, endlichen Automaten analog zur Arbeit von Julca-Aguilar und Hirata [13] in einen Offline-Datensatz konvertiert. Die erzielten Werte demonstrieren dabei die signifikante Verbesserung des damaligen State-of-the-Art der Offline-Erkennung von Diagrammskizzen. Zusätzlich be-

kräftigen die erreichten Erkennungsraten, dass ein Einsatz von Online-Datensätzen im Bereich des maschinellen Lernens darstellbar ist. Schäfer und Stuckenschmidt [16] sind aber ebenso der Ansicht, dass diese Art von Daten sich von realitätsnahen Offline-Skizzen unterscheiden.

Aus diesem Grund wird der in Abschnitt 2.3 bereits angesprochene Datensatz hdBPMN in mehreren Arbeiten [15, 16] von Schäfer und Stuckenschmidt präsentiert und eingesetzt. Dieser ist nach aktuellem Kenntnisstand der einzige Datensatz von Diagrammskizzen für maschinelle Lernverfahren, welcher ursprünglich für die Offline-Erkennung entstanden ist. Der Datensatz besteht dabei aus 502 Bildern und Annotationen handgezeichneter BPMN-Diagramme, welche mehr als 20.000 annotierte Diagrammelemente beinhalten. Diese sind im Zuge einer Lehrveranstaltung an der Universität Mannheim entstanden. Die meist fotografisch aufgenommenen Diagrammskizzen basieren auf freien Lösungen mehrerer benoteter Abgaben von 107 Studierenden. Die entstandenen Skizzen sind folglich nicht aus einer gegebenen Vorlage heraus entstanden, was wiederum zu einer höheren Varianz der Diagrammstruktur führt. Die im Datensatz enthaltenen Bilder variieren ebenso bezüglich des Diagrammhintergrunds, des Zeichengeräts und der Aufnahmequalität. All dies spricht für eine große Diagrammvielfältigkeit und erhöht die externe Validität der Skizzen. Die Annotationen sind nach der Datenerhebung manuell mit einem selbst implementierten Werkzeug zum Annotieren von BPMN-Diagrammen [33] getätigt worden. Diese wurden im Anschluss zusammen mit ihren zugehörigen Bilddateien mittels dem Skript *pybpmn* [34] in einen COCO-Datensatz (siehe Abschnitt 2.3) konvertiert. In der Arbeit *Sketch2BPMN* von Schäfer u. a. wird der Datensatz erstmalig eingesetzt, um das Lernverfahren Arrow R-CNN auf einem natürlichen Offline-Datensatz zu validieren. Hierbei werden zusätzliche Verfahren wie die der *Datenaugmentation* eingesetzt. Diese ermöglichen das Vergrößern kleinerer Datensätze durch das Anwenden mehrerer Bildfilter. Trotz zusätzlicher Herausforderungen natürlich entstandener Skizzen, welche ebenfalls in der zitierten Arbeit [15] illustriert werden, wird auch hier die automatisierte Erkennung dieser Diagramme mittels Arrow R-CNN durch starke Ergebnisse gewährleistet. Der Einsatz des maschinellen Lernverfahrens DiagramNet [16] bestätigt dies und verstärkt speziell bei Kanten die ohnehin schon starken Erkennungsraten auf dem hdBPMN-Datensatz.

Insgesamt zeigen speziell letztgenannte Arbeiten das enorme Potenzial von maschinellen Lernverfahren im Bereich der Skizzenerkennung. Ein zu hdBPMN analoger Offline-Datensatz von UML-Klassendiagrammen könnte die bereits erzielten Ergebnisse durch eine zuvor nicht gegebene Verfügbarkeit von Daten auf eine neue Diagrammdomäne übertragen. Dementsprechend orientiert sich der in dieser Arbeit präsentierte Datensatz in der Erhebung (siehe Kapitel 4), Annotation (siehe Abschnitt 5.2) und COCO-Konvertierung (siehe Abschnitt 5.3) an hdBPMN und wird in einem für die UML angepassten DiagramNet-Ansatz eingesetzt (siehe Abschnitt 6.2).

4. Erheben von UML-Klassendiagrammskizzen

Das Erheben von passenden Skizzen ist bei der Erstellung eines Offline-Datensatzes von handgezeichneten Diagrammen ein erster und essenzieller Schritt. Im Falle des Anwendungsgebiets dieser Bachelorarbeit, ist speziell die Gewinnung von möglichst natürlichen und realitätsnahen Klassendiagrammskizzen erstrebenswert. Damit geht das Auftreten von Eigenschaften einher, welche in Unterabschnitt 2.1.2 oder in verwandten Arbeiten mit ähnlichen Datensätzen [15, 16] illustriert wurden.

Die im Folgenden dargestellte Erhebung von Diagrammskizzen setzt sich aus zwei Quellen zusammen. Abschnitt 4.1 präsentiert dabei die erste und größte Datenquelle. Diese baut auf einer bewerteten Studierendenabgabe eines Lehrbetriebs am Karlsruher Institut für Technologie (KIT) auf. In diesem Schritt wird illustriert, wie die auf diese Art und Weise erhobenen Skizzen vor der Annotation aufbereitet wurden und welche Charakteristiken diese aufweisen. Die zweite, ergänzende Quelle handgezeichneter Klassendiagramme ist eine im Rahmen der Bachelorarbeit durchgeführte Nutzerstudie. In Abschnitt 4.2 wird der Aufbau dieser detailliert beschrieben. Des Weiteren werden die Eigenschaften der durch die Studie erhobenen Daten präsentiert. Um die Diagrammskizzen in ihrem Ursprung einzuordnen, werden ebenfalls demografische Informationen der Studienteilnehmer dargestellt. Zuletzt wird in Abschnitt 4.3 der Versuch beschrieben, den Lindholmen-Datensatz von Robles u. a. [31] (siehe Abschnitt 3.1) als dritte Datenquelle für den in dieser Arbeit präsentierten Datensatz zu nutzen. Im Zuge dessen werden dabei auftretende Probleme und Herausforderungen detailliert beschrieben.

4.1. Datenquelle 1: Studierendenaufgabe

Die für den in dieser Arbeit präsentierten Datensatz größte Quelle für Diagrammskizzen basiert auf einer bewerteten Studierendenabgabe des *Softwaretechnik 1*-Übungsbetriebs im Sommersemester 2021 am KIT. Diese fand am 19. Mai 2021 statt und ging somit dem offiziellen Start dieser Bachelorarbeit voraus. Die Studierenden hatten hier auf Grundlage einer Übung die Aufgabe, ein Klassendiagramm aus einer gegebenen Beschreibung eines Softwareprojektes händisch zu skizzieren. Im zugehörigen Aufgabentext wurden sie ebenso über die geplante Erhebung der Klassendiagramme für den Zweck dieser Abschlussarbeit informiert. Die Teilnahme und das Bereitstellen ihrer Lösung war dabei optional und konnte mit dem Anhängen einer Einverständniserklärung bestätigt werden. Insgesamt haben so 98 Studierende der Verwendung ihrer Lösung zum Zweck des Datensatzes zugestimmt und ihre Klassendiagrammskizze mitsamt Einwilligung in einem Bild- oder PDF-Format über ein Online-Portal hochgeladen.

Im Folgenden wird in Unterabschnitt 4.1.1 zunächst beschrieben, wie die so gewonnenen Daten in einem ersten Schritt bearbeitet und so für den Zweck des Datensatzes vorbereitet wurden. Im Zuge dessen wird ebenso illustriert, wann eine Diagrammskizze für das maschinelle Lernen nicht nutzbar und somit auch für einen Datensatz nicht geeignet sein kann. Anschließend werden in Unterabschnitt 4.1.2 die Eigenschaften der auf diesem Weg erhobenen Daten nach Bearbeitung dieser präsentiert. Hierbei wird ebenfalls Bezug auf die Anzahl spezifischer UML-Elementtypen innerhalb der Diagrammskizzen genommen.

4.1.1. Vorbereitung der Daten

Nach Abgabe aller 98 Lösungen, wurden zunächst alle als PDF-Dateien eingereichten Diagrammskizzen in ein JPG-Format konvertiert. In diesem Schritt wurden alle PDF-Dateien, welche mehr als eine Seite beinhalten, in mehrere Bilddateien getrennt. Mit den schon ursprünglich als Bild vorliegenden Diagrammen lagen nach dieser Konvertierung insgesamt 111 JPG-Dateien vor. Die größere Anzahl an Bilddateien im Vergleich zur Anzahl an Skizzierenden lässt sich am Aufbau der Übungsaufgabe, welche sich in zwei Teilaufgaben spaltet, begründen. Der erste Teil der Übung fordert dabei das Erstellen eines Klassendiagramms aus einer gegebenen Beschreibung heraus (siehe Abbildung A.4). Das hieraus entstandene Diagramm sollte anschließend in der zweiten Teilaufgabe gemäß einer ergänzenden Beschreibung erweitert werden (siehe Abbildung A.5). Die Separation der Aufgabenteile erfolgte hierbei von Fall zu Fall unterschiedlich. Einige Studierende haben ihre Lösungen farblich getrennt, das heißt die zweite Teilaufgabe befindet sich weiterhin auf dem gleichen Bild wie die erste. Andere haben hingegen ihre Lösungen auf zwei Bilddateien, beziehungsweise im Falle einer PDF-Datei auf zwei separate Seiten, aufgeteilt. Daraus resultiert das Existieren mehrerer Bilder pro Zeichner. Die Aufteilung auf mehrere Bilder, beziehungsweise Seiten, ist außerdem ein Mittel für das Darstellen sehr großer Diagramme. Ein einzelnes Bild enthält in diesen Fällen somit nur eine Teilmenge der im Diagramm enthaltenen Knoten und Kanten.

Der nächste Schritt zur Vorbereitung der Bilddateien für den Datensatz fokussiert sich auf die Anonymisierung der Daten. Initial wurden hierfür zunächst alle Metadaten der zugrundeliegenden Bilder gelöscht. Außerdem mussten ebenfalls einige Bildinhalte manuell bearbeitet werden. Da es sich bei den Skizzen um bewertete Übungsaufgaben einer Universität handelt, enthielten diese neben den handgezeichneten Diagrammen Aufgabenbeschreibungen, sowie Informationen wie Name und Matrikelnummer der jeweiligen Studierenden. Auch die zuvor beschriebene Einverständniserklärung ist folglich auf jeder Abgabe enthalten. All diese Informationen haben keinen Zusammenhang mit den dargestellten Diagrammskizzen und existieren ausschließlich aufgrund ihres Ursprungs. Die Daten sind daher ebenso ein Störfaktor für die spätere, maschinelle Erkennung von Diagrammen und passen folglich nicht in einen solchen Datensatz. Sowohl aus diesem, als auch aus dem Grund der Anonymisierung, wurden all die genannten Informationen manuell aus den Bildern entfernt.

Weiterhin ist das Auftreten mehrerer Bilder von identischen Zeichnern zu betrachten. Die Spaltung in verschiedene Dateien erfolgt insbesondere bei der Separierung unterschiedlicher Aufgabenteile. Das Diagramm der zweiten Teilaufgabe ist dabei in den meisten Fällen eine Obermenge des dargestellten Diagramms des ersten Aufgabenteils. In diesen

Fällen wurde das Bild entfernt, welches nur eine Teilmenge des gesamten Diagramms visualisiert. Dieser Schritt soll die Validität des Datensatzes erhöhen, da eine Erkennung bereits trainierter Diagrammelemente nicht aussagekräftig ist. Des Weiteren wurden alle zusammenhängenden Skizzen, welche ursprünglich auf mehrere Seiten getrennt wurden, in einer einzigen Bilddatei vereinigt. Zusätzlich wurden drei weitere Bilder, welche nur eine kleine Untermenge eines Diagramms visualisieren und bei welchen eine Vereinigung aufgrund von Ungenauigkeiten der Zeichner nicht ohne Weiteres darstellbar gewesen ist, entfernt. Alle aus dieser Quelle gewonnenen Diagrammskizzen stammen somit von jeweils unterschiedlichen Zeichnern. Dies verstärkt wiederum die externe Validität späterer Erkennungsraten auf Basis dieses Datensatzes, da alle enthaltenen Daten unabhängig voneinander sind. Diese Unabhängigkeit äußert sich unter anderem an großen Variationen bezüglich Modellierungsstil, Aufnahmequalität, Diagrammhintergrund oder dem zugehörigen Zeichengerät. Diese Punkte variieren ebenso im hdBPMN-Datensatz und werden in der zum DiagramNet zugehörigen Arbeit [16] illustriert.

In einem nächsten Schritt wurden alle 98 Diagrammskizzen manuell auf Besonderheiten und die Anzahl der vorkommenden Elemente analysiert. Zeitgleich wurden *unnutzbare Daten* entfernt. Hintergrund hierbei ist die Annotation der im Datensatz enthaltenen Diagramme, welche für ein maschinelles Lernverfahren essenziell ist. Unnutzbar kann eine Skizze sein, wenn es für einen Menschen nicht darstellbar ist, diese zu annotieren. Begründbar ist dies unter anderem durch eine sehr schlechte Aufnahmequalität und Auflösung. Im Rahmen dieser Arbeit wurde eine unnutzbare Bilddatei identifiziert und anschließend entfernt (siehe Abbildung 4.1).

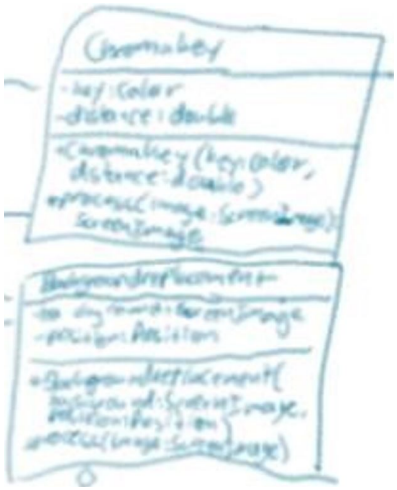
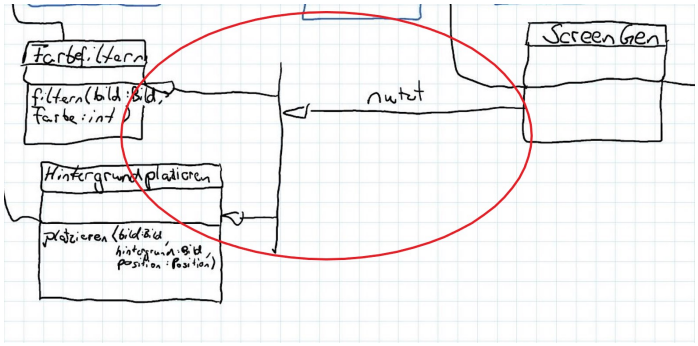


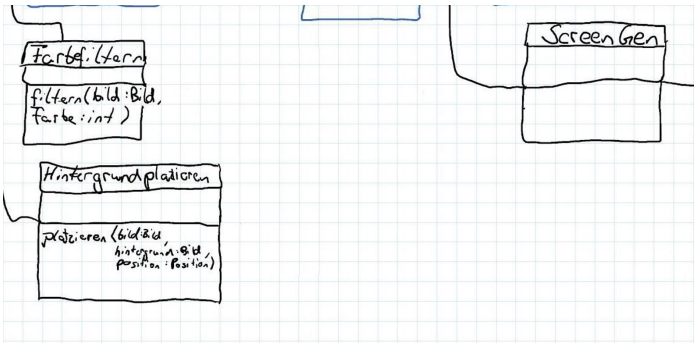
Abbildung 4.1.: Ausschnitt einer UML-Klassendiagrammskizze, welche aufgrund ihrer Auflösung und der daraus resultierenden Nichtidentifizierbarkeit der Textblöcke für den Datensatz unbrauchbar ist

Zusätzlich wurden Diagrammelemente identifiziert, welche in ihrer Gestalt nicht Bestandteil der UML-Klassendiagrammsyntax sind. Das Annotieren solcher Elemente ist dementsprechend nicht darstellbar. Bei einem Einsatz dieser Daten für ein maschinelles Lernverfahren resultieren daraus zwangsläufig verfälschte Ergebnisse, da eine Erkennung eines nicht annotierten, aber dennoch dargestellten Elements, zwangsläufig zu einem

falsch positiven Resultat führt. Ein Enthalten dieser falschen Elemente in einem Datensatz für maschinelle Lernverfahren ist aus diesem Grund nicht sinnvoll. Die restlichen Bestandteile der zugehörigen Diagramme können hingegen problemlos annotiert und für ein maschinelles Lernverfahren verwendet werden. Die dargestellte Problematik wurde daher durch das Ausschneiden dieser Elemente gelöst. Eine solche Bearbeitung eines Ausgangsbildes ist im Zuge dieser Arbeit bei sieben Skizzen erfolgt. Die getätigten Änderungen wurden aus Transparenzgründen dokumentiert. Des Weiteren wurden neben denen schlussendlich im Datensatz enthaltenen Klassendiagrammskizzen ebenso deren Ausgangsbilder gesichert. Zukünftige Arbeiten mit dem präsentierten Datensatz sind deshalb weiterhin in der Lage, den Umgang mit Fehlern dieser Art selbstständig nach ihrem Anwendungszweck zu entscheiden. Abbildung 4.2 illustriert das Ausschneiden eines syntaktisch falschen Elements grafisch.



(a)



(b)

Abbildung 4.2.: UML-Klassendiagrammskizze mit syntaktisch falschem Element (rot umrandet) **(a)** vor und **(b)** nach der in Unterabschnitt 4.1.1 beschriebenen Bildbearbeitung

In einem letzten Schritt wurden alle 97 Diagrammskizzen gemäß der Konvention *ex_<ID>* umbenannt. Die ID entspricht dabei einer eindeutigen Zahl zwischen 1 und 97 und wurde den zugehörigen Bilddateien willkürlich zugeordnet.

4.1.2. Eigenschaften der Daten

Die auf diese Art und Weise erhobenen UML-Klassendiagramme bringen einige Eigenschaften und Auffälligkeiten mit sich, die im Folgenden dargestellt werden.

Jede der Diagrammskizzen wurde frei von Studierenden auf ein Blatt Papier oder digital mithilfe eines Tables händisch gezeichnet. Das bedeutet, dass die Zeichner in keinster Weise an eine Vorlage gebunden waren, wie beispielsweise beim Datensatz von Awal u. a. [32]. Dies äußert sich unter anderem an für Skizzen typischen Ungenauigkeiten, wie das unvollständige Zeichnen von Formen oder das Durchstreichen von Elementen (siehe Abbildung 2.6). Des Weiteren folgt durch das freie Zeichnen eine große Vielfältigkeit der Diagrammstruktur. Diese Strukturunterschiede lassen sich unter anderem durch große Abweichungen der Diagrammelementhäufigkeiten bestätigen. Tabelle 4.1 verdeutlicht dies mit einer Statistik zu den in den Skizzen auftretenden Knoten, Kanten und Textblöcken. Insbesondere die Standardabweichung und die Differenz zwischen Maximum und Minimum aller Elemente verdeutlichen große Abweichungen in der Darstellung der exakt gleichen Aufgabenstellung.

Typ	**Knoten**	**Kanten**	**Textblöcke**
Gesamtanzahl	1397	1522	4941
Mittelwert	14,4	15,7	50,9
Minimum	5	0	22
Maximum	26	34	101
Standardabweichung	4,1	6,0	17,1

Tabelle 4.1.: Studierendenabgabe: Knoten, Kanten und Textblöcke

Die auftretenden Kantenkategorien umfassen hier exakt die in Abbildung 2.4 dargestellten Kantentypen und die Textblöcke analog die in Unterabschnitt 2.1.1 eingeführten Arten von Text. Bei den Knoten wird, zusätzlich zu den in Unterabschnitt 2.1.1 genannten Knotentypen, das Rechteck des Qualifizierers inkludiert. Diese Zuteilung des eigentlichen Bestandteils einer Kante, soll so die Separation aller auftretenden Kantenarten erleichtern. Des Weiteren werden die Elementtypen *Bibliothek* und *Utility* als separate Knoten gezählt. Diese unterscheiden sich von ihrem Modellierungsverhalten und ihrer Form nur sehr geringfügig von Klassen und werden in der Regel nicht als eigenständiges Element betrachtet. Aufgrund der gegebenen Aufgabenstellung haben einige Studierende diese jedoch analog zu Enumeration und Schnittstelle mit einer vorhergehenden Textbeschriftung gekennzeichnet und werden dementsprechend ebenso separiert. Tabelle 4.2 zeigt alle spezifischen Elementtypen mit deren Häufigkeit.

Durch das freie Skizzieren in der Entstehung und der Unabhängigkeit aller Diagramme folgt ebenfalls eine große Varianz bezüglich Diagrammhintergrund, Aufnahmequalität und Zeichengerät. Ersteres äußert sich am zugehörigen Papiertyp der Zeichnung oder dem Hintergrund der digitalen Anwendung des genutzten Tablets. Dieser Hintergrund ist in den meisten Fällen der zugrundeliegenden Daten kariert (49) oder blank (39). Die restlichen Diagramme werden auf einem punktierten (6) oder linierten Hintergrund (3) dargestellt. Wie bereits in Unterabschnitt 2.1.2 illustriert, bringen vor allem karierte und

Typ	Elementart (Häufigkeit)
Knoten	Klasse (1050), Schnittstelle (87), Enumeration (96), Abstrakte Klasse (106), Qualifizierer (9), Paket (5), Raute aus mehrgliedriger Assoziation (5), Notiz (13), Objekt (15), Utility (6), Bibliothek (5)
Kanten	Gerichtete Assoziation (242), Ungerichtete Assoziation (189), Aggregation (153), Komposition (147), Abhängigkeit (116), Vererbung (475), Realisierung (186), Notizverbindung (14)
Textblöcke	Elementname (1382), Attribut (859), Methode (825), Enumerationswert (335), Kantenbeschriftung (653), Multiplizität (872), Kommentar (Text in Notizknoten) (15)

Tabelle 4.2.: Studierendenabgabe: Häufigkeiten spezifischer Elementarten (detaillierte Statistik siehe Tabelle A.1)

linierte Papierarten weitere Herausforderungen bei der Erkennung relevanter Elementformen mit sich. Die hier zusätzlich vorhandenen Linien müssen dabei von den tatsächlich gezeichneten Linien unterschieden werden. Wie der hdBPMN-Datensatz [15, 16] schon zeigt, ist daher eine Präsenz und Variation verschiedener Hintergründe für das Training eines maschinellen Lernverfahrens sinnvoll. Diese Vielfältigkeit ist in den hier gewonnenen Daten somit gegeben.

Die Varianz bezüglich der Aufnahmequalität äußert sich in den meisten Fällen durch die Art der Bildsicherung. Unterscheiden kann man hier zwischen *eingescannten* (65) und *fotografierten* (32) Lösungen. Letztere bringen dabei zusätzliche Herausforderungen mit sich, da das zugehörige Bild zusätzliche Objekte wie beispielsweise einen Tisch enthalten kann (siehe Abbildung 4.3). Digital entstandene Diagrammbilder, welche auf einem Tablet händisch gezeichnet wurden, unterscheiden sich in ihrer Aufnahmequalität nur geringfügig von eingescannten Lösungen und werden daher nicht von diesen separiert.

Das verwendete Zeichengerät kann in den hier gewonnenen Daten selbst innerhalb eines Diagramms variieren. Dies lässt sich wiederum durch die Separation zweier Teilaufgaben und die zugehörige Aufgabenstellung erklären (siehe Abbildung A.5). In diesen Fällen werden verschiedene Elementformen mit unterschiedlichen Farben dargestellt. Die gezeichneten Linien können sich dabei zusätzlich bezüglich ihrer Dicke unterscheiden. Außerdem ist eine weitere Variation des Zeichengeräts zwischen Text und Elementlinien möglich. Auch zwischen verschiedenen Skizzen besteht diesbezüglich eine große Vielfältigkeit. Zu den am häufigsten verwendeten Zeichengeräte zählen *Kugelschreiber* und *Fineliner*. Digital händisch gezeichnete Diagramme sind dabei eingescannten, von Fineliner gezeichneten Skizzen sehr ähnlich und werden folglich nicht von diesen separiert. Des Weiteren existieren in den zugrundeliegenden Daten wenige Zeichnungen mit *Bleistift* und *Buntstift*. Exakte Häufigkeiten können hier nicht angegeben werden, da die Zeichner darüber keine Auskunft gegeben haben und eine nachträgliche Identifikation des Zeichengeräts nur bedingt möglich und ungenau ist.

Weitere Auffälligkeiten in diesen Diagrammskizzen sind häufig auftretende, syntaktische Fehler. Diese treten in verschiedensten Formen in rund ein Drittel aller Lösungen

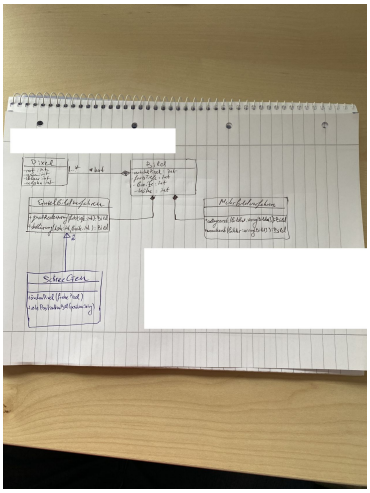


Abbildung 4.3.: Darstellung einer fotografierten Lösung der Studierendenabgabe mitsamt Hintergrund. Name, Matrikelnummer und Einverständniserklärung wurden mit einem weißen Balken überdeckt.

auf. Dieser hohe Anteil an fehlerhaften Diagrammen lässt sich höchstwahrscheinlich mit der geringen UML-Erfahrung der Zeichner begründen. Die zugehörige Lehrveranstaltung wird in der Regel im 2. Fachsemester besucht und ist für viele Studierende der erste Berührungspunkt mit UML-Klassendiagrammen. Dies äußert sich vor allem an Fehlern, in welchen eine falschen Kantenart verwendet wurde. Die Skizzierenden haben in diesen Fällen beispielweise eine Klasse und eine Schnittstellen mittels einer Vererbungskante verbunden (siehe Abbildung 4.4), was laut der offiziellen UML-Syntax [4] nicht zulässig ist. Fehler dieser Art wurden aber im Gegensatz zu nicht existenten Elementformen (siehe Abbildung 4.2) in den Daten belassen, da in erster Linie vor allem die geometrische Form für eine Objekterkennung ausschlaggebend ist und weniger syntaktische Verbindungsregeln. Alle identifizierten Regelbrüche wurden dabei dokumentiert, so dass mit diesen in zukünftigen Betrachtungen gegebenenfalls in einer anderen Weise umgegangen werden kann. Im hdBPMN-Datensatz von Schäfer u. a. [15] wurden ähnliche BPMN-Modellierungsfehler analog zum Vorgehen dieser Arbeit im finalen Datensatz belassen.

4.2. Datenquelle 2: Online-Nutzerstudie

Das Nutzen mehrerer Datenquellen kann sowohl die Größe als auch die Vielfältigkeit eines Datensatzes erhöhen. Die im Folgenden beschriebene Online-Nutzerstudie wurde vom 16.09.2021 bis zum 06.10.2021 durchgeführt und hatte das Ziel, die in Abschnitt 4.1 beschriebenen Diagramme durch eine ergänzende Datenerhebung zu erweitern. In Unterabschnitt 4.2.1 wird dabei detailliert der Studienaufbau dargestellt. In diesem Schritt werden die für die Erhebung von Skizzen eingesetzten Aufgabenszenarien, die Art und Weise des zugrundeliegenden Samplings und die tatsächliche Durchführung der Fallstudie präsentiert. In Unterabschnitt 4.2.2 werden anschließend die durch die Studie gewonnenen Diagrammskizzen analog zu Unterabschnitt 4.1.2 analysiert. Des Weiteren werden zusätz-

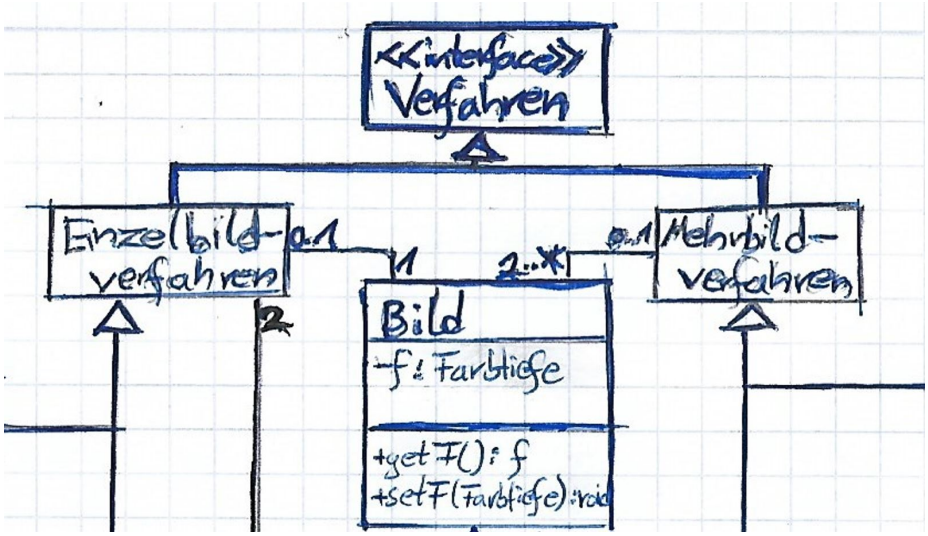


Abbildung 4.4.: Ausschnitt einer Diagrammskizze aus den Studierendenabgaben. Die zwei oberen Klassen sind dabei über Vererbungskanten mit einer Schnittstelle verbunden.

lich erhobene, demografische Daten der Skizzierenden präsentiert, um den Ursprung der UML-Klassendiagrammskizzen einzuordnen.

4.2.1. Aufbau und Durchführung

Studienaufbau Das Ziel des im Folgenden beschriebenen Studienaufbaus ist die Erhebung möglichst vieler, händisch gezeichneter UML-Klassendiagramme. Die Studie wurde hierfür komplett online durchgeführt. Eine Durchführung dieser Art hat dabei den Vorteil, dass die Teilnehmer von ihrem normalen Arbeitsplatz teilnehmen können und somit keine Distanz für das Teilnehmen an der Studie zurücklegen müssen. Des Weiteren ist die beschriebene Datenerhebung asynchron verlaufen. Dies bedeutet, dass die Studienteilnehmer die Studie selbstständig ohne Betreuer zu einem beliebigen Zeitpunkt durchführen konnten. Die zwei genannten Faktoren machen den Prozess daher deutlich unkomplizierter und weniger zeitintensiv für jeden Probanden. Gleichen Zweck hatte auch die Wahl der konkreten Aufgabenstellung. Diese sollte, analog zu der Studierendenabgabe der vorausgehenden Datenerhebung, das Skizzieren exakt eines Klassendiagramms aus einem gegebenen Szenario heraus beanspruchen. Das Fordern mehrerer UML-Klassendiagrammskizzen pro Person könnte zu viel Zeit beanspruchen und potenzielle Studienteilnehmer abschrecken. Der im Szenario beschriebene Sachverhalt sollte außerdem aus identischen Gründen von geringerer Komplexität und somit möglichst für eine große Teilnehmergruppe verständlich sein. Aus diesem Grund wurden drei Szenarien bestehender Softwaretechnik 1-Klausuren des KIT für den Zweck der Datenerhebung wiederverwendet (siehe Abbildung A.1, Abbildung A.2 und Abbildung A.3). Diese wurden leicht verändert, um die Komplexität und Länge aller Aufgaben auf ein ungefähr gleiches Level zu heben (siehe Abbildung A.9, Abbildung A.10 und Abbildung A.11). Die Änderungen umfassen hierbei das Verändern und Entfernen bestehender und das Hinzufügen neuer Sätze. Die Szenarien beschreiben

dabei jeweils Situationen, welche kein bestimmtes Domänenwissen für ihr Verständnis voraussetzen. Des Weiteren sind sie mit einer Länge von fünf Sätzen nicht zu umfangreich. Es existieren innerhalb der Aufgabenstellung zudem keine Vorgaben bezüglich der zu verwendenden Diagrammelemente oder deren Struktur. Die Probanden waren daher frei in ihrer Modellierung und konnten somit ein möglichst natürliches UML-Klassendiagramm skizzieren. Die hierfür verwendeten, verschiedenen Szenarien sollten dabei möglichst gleichmäßig auf alle Studienteilnehmer verteilt werden. Diese Variation hatte das Ziel, die Diversität der entstehenden Diagrammstrukturen und somit die externe Validität der Daten zu erhöhen.

Die Verteilung der Szenarien wurde durch das Erstellen dreier unterschiedlicher *Google Formulare* gewährleistet, welche sich ausschließlich am gegebenen Szenario unterscheiden (siehe Abbildung A.6 bis Abbildung A.11). Diese wurden über einen von einem *Google Apps Skript* erzeugten Link zufällig und möglichst gleich verteilt aufgerufen. Um den Ursprung der Daten einzuordnen, wurden in der zugehörigen Umfrage außerdem demografische Daten wie Alter, Geschlecht, Bildungsabschluss, aktuelle Tätigkeit und Erfahrung mit UML-Klassendiagrammen abgefragt. Nach Angabe dieser wurden die Probanden aufgefordert, ihr Szenario händisch auf ein Blatt Papier zu zeichnen. Des Weiteren wurden sie gebeten, die von ihnen verwendeten UML-Elemente möglichst korrekt einzusetzen und sich weniger auf eine semantisch korrekte Repräsentation des Szenarios zu fokussieren. Die entstandene Lösung sollte im Anschluss in einen von *bwSync & Share* geteilten Ordner in einem PDF- oder Bildformat hochgeladen werden. Zusätzlich konnte durch eine nicht verpflichtende Angabe einer E-Mail-Adresse nach Studienabschluss optional an einem Gewinnspiel teilgenommen werden.

Sampling Die Zielgruppe der Studie sind Personen, welche eine gewisse Erfahrung mit UML-Klassendiagrammen besitzen. Um diese zu erreichen, wurden über eine willkürliche Stichprobe vor allem verschiedene Studierendenkanäle des KIT angesprochen. Diese beinhalten primär Mailinglisten, auf denen zum Großteil Studierende der Informatik Adressat sind. Des Weiteren wurden zusätzlich Studierende über weitere Internetmedien erreicht. Um potenziell für eine größere Teilnehmerzahl zu sorgen, wurden zudem unter allen Teilnehmenden zwei 10 Euro *Amazon*-Gutscheine verlost. Jeder Studienteilnehmer hatte nach Kenntnisnahme bis zum 06.10.2021 Zeit, eine selbst gezeichnete UML-Klassendiagrammskizze einzureichen.

Durchführung Die Nutzerstudie zur Datenerhebung wurde vom 16.09.2021 bis zum 06.10.2021 durchgeführt. Innerhalb dieses Zeitraums haben insgesamt 18 verschiedene Probanden ein Bild eines handgezeichneten UML-Klassendiagramms hochgeladen. Mit all diesen Skizzen wurde im Anschluss analog zu Unterabschnitt 4.1.1 verfahren. Hierbei konnten keine unnutzbaren Skizzen oder syntaktisch nicht existenten Elemente identifiziert werden. Alle Bilddateien wurden im Anschluss nach dem Muster *us_<ID>* umbenannt, wobei ID eine Zahl zwischen 1 und 18 repräsentiert. Die 18 Lösungen verteilen sich dabei auf die drei gegebenen Szenarien relativ gleichmäßig (siehe Tabelle 4.3).

Eine exakte Gleichverteilung konnte durch den bei einer asynchronen Studiendurchführung zwangsläufig gegebenen Zufallsfaktor nicht erreicht werden. Wie bereits Un-

Szenario	Häufigkeit
Szenario 1: „Schiff“	5
Szenario 2: „Güterzug“	7
Szenario 3: „Fachwerkhaus“	6

Tabelle 4.3.: Nutzerstudie: Verteilung der Probanden auf die drei gegebenen Szenarien (siehe Abbildung A.9, Abbildung A.10 und Abbildung A.11)

terabschnitt 4.1.2 aufgezeigt hat, kann eine große Vielfältigkeit verschiedenster Skizzenmerkmale auch innerhalb eines einzelnen Szenarios gegeben sein. Die Ungleichheit der Diagrammverteilung stellt daher keineswegs die Validität und Datenvarianz in Frage.

4.2.2. Ergebnisse

Im Folgenden wird in einem ersten Schritt Bezug auf die im Zuge der Umfrage erhobenen Teilnehmerdaten genommen. Mit diesen lassen sich die im Anschluss analysierten UML-Klassendiagramme in ihrem Ursprung einordnen.

Bei der Geschlechterverteilung der Probanden und Probandinnen überwiegt der Anteil männlicher Teilnehmer (siehe Abbildung 4.5). Für diese ist jedoch keinerlei Korrelation mit einem bestimmten Modellierungsstil von UML-Klassendiagrammen zu erwarten.

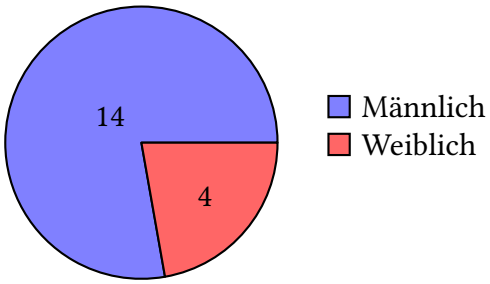


Abbildung 4.5.: Geschlechterverteilung der Nutzerstudie

Alle 18 Studienteilnehmer und -teilnehmerinnen lassen sich zudem in die Altersgruppe von 18 bis 29 Jahren einordnen. Der Großteil befindet sich dabei derzeit in einem aktiven Studium im Bereich der Informationstechnik (siehe Abbildung 4.6).

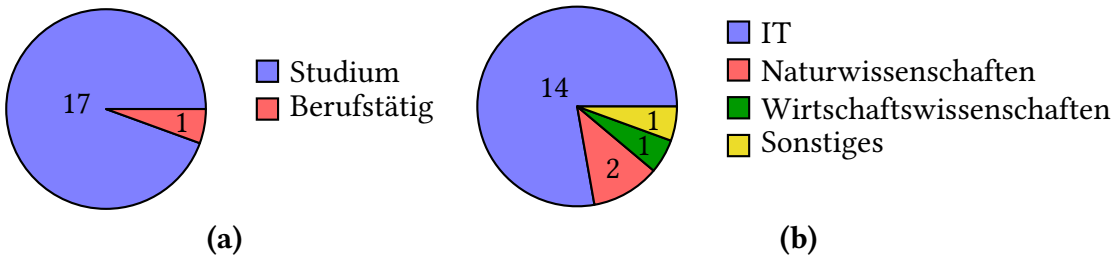


Abbildung 4.6.: **(a)** Derzeitige Tätigkeit und **(b)** dessen Bereich aller Studienteilnehmer

Dies lässt sich mit der beschriebenen Methode des Samplings erklären, welche sich auf Informatikstudierende des KIT fokussiert hat. Die durch die Nutzerstudie erhobenen Diagrammskizzen sind also analog zu den in Abschnitt 4.1 beschriebenen Daten hauptsächlich von Studierenden angefertigt worden. Der höchste Bildungsabschluss aller Probanden beschränkt sich folglich auf eine Hochschulreife oder ein abgeschlossenes Studium (siehe Abbildung 4.7).

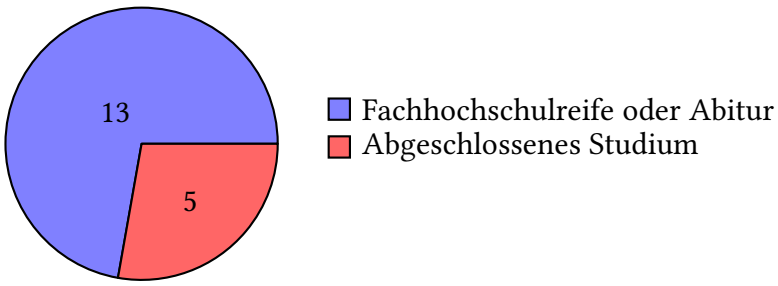


Abbildung 4.7.: Höchster Bildungsabschluss aller Studienteilnehmer

Ein für die Einordnung der Daten ebenfalls interessanter Wert ist die im Zuge der Studie abgefragte Selbsteinschätzung bezüglich der Erfahrung mit dem Skizzieren von UML-Klassendiagrammen. Diese sollte mit einem Zahlenwert zwischen 1 (keine Erfahrung) und 5 (sehr erfahren) angegeben werden (siehe Tabelle 4.4). Der hieraus resultierende Mittelwert beträgt dabei ungefähr 3,1. Dies spricht für eine mittlere Erfahrung mit UML-Klassendiagrammen über alle Teilnehmenden hinweg.

Geschätzte Erfahrung mit UML-Klassendiagrammen	**Häufigkeit**
1 (keine Erfahrung)	0
2	5
3	7
4	5
5 (sehr erfahren)	1

Tabelle 4.4.: Selbsteinschätzung aller Studienteilnehmer bezüglich ihrer Erfahrung mit dem Skizzieren von UML-Klassendiagrammen

Bei einer zusätzlichen Freitextfrage wurde zudem der zugehörige Kontext abgefragt, in welchem die Studienteilnehmer diese UML-Erfahrung gesammelt haben. Bis auf eine Ausnahme ohne Antwort, hat hierbei jeder Teilnehmer angegeben, UML-Klassendiagramme im Laufe ihres Studiums behandelt zu haben. Zusätzlich haben sich zwei weitere Probanden oder Probandinnen im Rahmen ihrer Arbeit mit dem Modellieren von UML-Klassendiagrammen beschäftigt. Insgesamt zeigen alle vorgestellten Nutzerdaten, dass die hier erhobenen Klassendiagramme in ihrem Ursprung stark vergleichbar mit denen der in Abschnitt 4.1 beschriebenen Daten sind. Beide Diagrammquellen decken folglich zum Großteil UML-Klassendiagramme durchschnittlich erfahrener Zeichner ab, welche diese zuvor hauptsächlich im Kontext der Lehre gezeichnet haben.

Die im Zuge der Nutzerstudie erhobenen Klassendiagrammskizzen unterscheiden sich dennoch von den Daten aus Abschnitt 4.1 bezüglich mehrerer Faktoren. Wie bereits in Unterabschnitt 4.2.1 illustriert, wurden für die Studie bewusst weniger komplexe Szenarien gewählt. Folglich sind auch die dadurch entstandenen Diagramme weniger umfangreich. Tabelle 4.5 zeigt dies anhand der Häufigkeiten der in den Klassendiagrammskizzen auftretenden Knoten, Kanten und Textblöcken.

Typ	Knoten	Kanten	Textblöcke
Gesamtanzahl	156	155	394
Mittelwert	8,7	8,6	21,9
Minimum	3	2	12
Maximum	11	13	37
Standardabweichung	1,9	2,8	6,6

Tabelle 4.5.: Nutzerstudie: Knoten, Kanten und Textblöcke

Die durch die Studie gewonnenen Skizzen besitzen erheblich weniger Diagrammelemente als die Daten der Studierendenabgabe (siehe Tabelle 4.1). Daraus resultiert wiederum eine größere Vielfältigkeit der Diagrammstrukturen des Gesamtdatensatzes. Des Weiteren existiert ebenfalls innerhalb der durch die Nutzerstudie erhobenen Klassendiagramme eine Varianz bezüglich der Anzahl der verwendeten Elemente. Die spezifisch genutzten Knotentypen fokussieren sich hierbei speziell auf Klassen. Dies ist ebenso durch die weniger komplexen Szenarien erklärbar. Auch die verwendeten Kanten- und Textarten sind aus gleichem Grund eine Untermenge der auftretenden Elemente in den von Abschnitt 4.1 beschriebenen Daten. Tabelle 4.6 zeigt dabei die in den Nutzerstudiendaten auftretenden, spezifischen UML-Elemente mit deren Häufigkeiten.

Typ	Elementart (Häufigkeit)
Knoten	Klasse (154), Schnittstelle (1), Abstrakte Klasse (1)
Kanten	Gerichtete Assoziation (11), Ungerichtete Assoziation (34), Aggregation (25), Komposition (24), Abhängigkeit (5), Vererbung (54), Realisierung (2)
Textblöcke	Elementname (156), Attribut (75), Methode (19), Kantenbeschriftung (22), Multiplizität (122)

Tabelle 4.6.: Nutzerstudie: Häufigkeiten spezifischer Elementarten (detaillierte Statistik siehe Tabelle A.2)

Analog zu Unterabschnitt 4.1.2, existieren auch innerhalb dieser Diagrammskizzen Unterschiede bezüglich Diagrammhintergrund, Aufnahmequalität und Zeichengerät. Ersteres beschränkt sich hier auf karierte (6) und blanke (12) Papiertypen. Im Gegensatz zur ersten Datenquelle sind die Lösungen häufiger fotografiert (12) als eingescannt (6) worden. Ebenso wie in Unterabschnitt 4.1.2 werden digital händisch gezeichnete Diagramme nicht von eingescannten Skizzen unterschieden. Dies gilt auch für das Zeichengerät, welches bei einer auf einem Tablet entstandener Lösung sehr einem Fineliner ähnelt. Dieser ist nach

dem Kugelschreiber das am häufigsten verwendete Zeichengerät in den zugrundeliegenden Daten. In wenigen Fällen wurden außerdem Bleistifte für das Zeichnen der Diagramme verwendet. Im Gegensatz zu den Skizzen aus Abschnitt 4.1, variiert das Zeichengerät hier ausschließlich zwischen verschiedenen Lösungen. Begründbar ist dies durch die Tatsache, dass die Aufgabenstellung nicht in zwei Teilaufgaben getrennt wurde.

Das Auftreten syntaktischer Fehler ist auch in vier dieser Diagrammskizzen gegeben. Dies äußert sich ebenfalls am Verwenden falscher Kantenarten (siehe Abbildung 4.8). Alle auftretenden Fehler wurden dabei in den Daten belassen und dokumentiert.

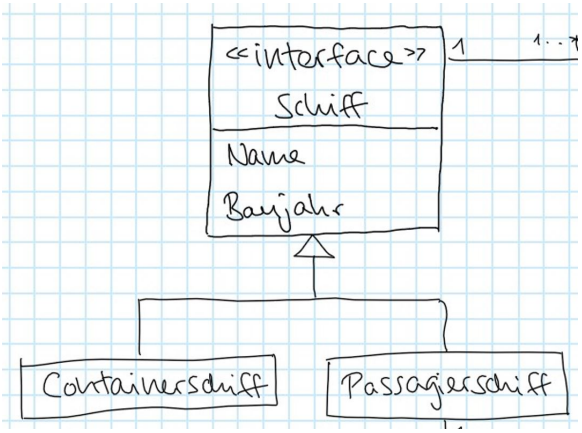


Abbildung 4.8.: Ausschnitt einer Diagrammskizze, welche im Rahmen der Nutzerstudie erhoben wurde. Zwei Klassen sind, ähnlich wie in Abbildung 4.4, über Vererbungskanten mit einer Schnittstelle verbunden.

Insgesamt weisen die im Zuge der Studie erhobenen UML-Klassendiagramme typische Eigenschaften von Skizzen auf (siehe Unterabschnitt 2.1.2). Dies macht sie daher für den in dieser Arbeit präsentierten Datensatz geeignet. Die ebenfalls illustrierten Unterschiede zur ersten Datenquelle führen außerdem zu einer größeren Diversität des Gesamtdatensatzes (siehe Tabelle A.3).

4.3. Lindholmen-Datensatz

Der in Abschnitt 3.1 vorgestellte Lindholmen-Datensatz von Robles u. a. [31] besitzt Verweise auf in GitHub-Projekten gesicherte UML-Diagramme verschiedenster Art. Inkludiert sind hier ebenfalls handgezeichnete UML-Klassendiagramme. Aus diesem Grund ist in dieser Arbeit der Lindholmen-Datensatz als dritte Quelle von Skizzen in Betracht gezogen worden. Im Folgenden wird der Versuch beschrieben, manuell aus diesem handgezeichnete UML-Klassendiagramme zu erheben.

Der Lindholmen-Datensatz besteht aus mehreren Datenbanktabellen. Hierbei werden unter anderem Informationen über das jeweilige GitHub-Projekt oder die darin enthaltenen UML-Klassendiagramme gespeichert. Für den Kontext dieser Arbeit sind dabei ausschließlich handgezeichnete Klassendiagramme von Relevanz. Ein Eintrag über die Entstehungsart der UML-Klassendiagramme existiert dabei nicht. Das alleinige Filtern

nach Skizzen ist somit nicht ohne Weiteres darstellbar. Eine manuelle Betrachtung vorhandener Diagrammbilder ist somit für das Finden von handgezeichneten Klassendiagrammen notwendig.

Eine weitere Herausforderung stellen die zugehörigen GitHub-Repositories dar. Nicht jedes dieser ist zum derzeitigen Zeitpunkt noch öffentlich einsehbar. Auch einzelne UML-Dateien innerhalb dieser Projekte können im Laufe der Zeit entfernt worden sein. Des Weiteren ist ein öffentliches GitHub-Projekt keineswegs automatisch *Open-Source*. Dies verhindert das freie Nutzen der Diagramme für einen neuen Anwendungszweck, da die Bilder standardmäßig urheberrechtlich geschützt sind. Aus diesem Grund ist zusätzlich eine manuelle Betrachtung der zu den GitHub-Projekten zugehörigen Lizenzen notwendig, falls diese denn überhaupt existieren.

Im Zuge der Bachelorarbeit wurde dennoch der Versuch unternommen, handgezeichnete UML-Klassendiagramme mithilfe der besagten Datenbank zu gewinnen. Zu diesem Zweck wurden vorhandene Bilddaten nach ihrem Dateinamen gefiltert. Dieser sollte dabei verschiedene Schlüsselworte enthalten, welche auf ein händisch gezeichnetes Diagramm deuten könnten. Die daraus resultierenden Ergebnisse sollten anschließend mit ihrem zugehörigen GitHub-Repository manuell bewertet werden. Insbesondere wurde der Erhalt der Teilstrings *„Sketch“* und *„Drawing“* geprüft.

Insgesamt wurden 36 Verweise auf Bilddateien identifiziert, welche „Sketch“ in ihrem Dateinamen beinhalteten. Davon waren sieben Dateien bereits nicht mehr öffentlich aufrufbar. Zehn weitere der Bilder stellten dabei kein UML-Klassendiagramm dar, während 16 der Ergebnisse auf CASE-Tool generierte Klassendiagramme verwiesen. Lediglich drei der Resultate verwiesen auf eine Klassendiagrammskizze. Die zugehörigen GitHub-Repositories enthielten jedoch alle keine Open-Source-Lizenz.

Weitere 16 Ergebnisse wurden mit dem Teilstring „Drawing“ erkannt. Von diesen verwiesen zehn auf von CASE-Tool erzeugte Klassendiagramme, sowie fünf auf Bilddateien ohne UML-Klassendiagramme. Eine der Dateien war wiederum nicht mehr öffentlich verfügbar.

Eine manuelle Betrachtung aller Klassendiagrammbilder mit deren zugehörigen GitHub-Repositories könnte unter Umständen zur Gewinnung geeigneter Skizzen führen. Dies wäre jedoch mit einem enormen Aufwand verbunden. Aus diesem Grund wurde in dieser Arbeit von einer weiteren Betrachtung des Lindholmen-Datensatzes abgesehen. Eine weitere und intensivere Untersuchung dieses Datensatzes könnte dennoch in zukünftigen Arbeiten ein Mittel zur Erhebung von UML-Klassendiagrammskizzen sein.

5. Erstellen eines Datensatzes für maschinelle Lernverfahren

Um einen Datensatz für den Einsatz maschineller Lernverfahren zu erstellen, müssen die in Kapitel 4 erhobenen UML-Klassendiagrammskizzen weiter aufbereitet werden. Das Annotieren der in den Daten enthaltenen Diagrammelemente ist hierbei ein erster Schritt. Relevante Knoten, Kanten und Textblöcke müssen in diesem Schritt sowohl klassifiziert als auch mit einer passenden Bounding-Box versehen werden. Zu diesem Zweck wird im Rahmen der Bachelorarbeit ein bestehendes Softwarewerkzeug zur Annotation von BPMN-Elementen auf die Domäne der UML-Klassendiagramme erweitert. Abschnitt 5.1 stellt hierfür den *UML-Image-Annotator* [35] vor, welcher die präzise Datenannotation von UML-Klassendiagrammen ermöglicht. Anschließend wird in Abschnitt 5.2 der explizite Einsatz des Werkzeugs an den Skizzen aus Kapitel 4 beschrieben. Zusätzlich ist eine Transformation aller Bilddateien mit den zugehörigen Annotationen in das COCO-Datenformat (siehe Abschnitt 2.3) notwendig. Diese Konvertierung wird in Abschnitt 5.3 dargestellt.

5.1. Aufbau und Funktionalität des UML-Image-Annotators

Der *BPMN-Image-Annotator* [33] ermöglicht das Annotieren von BPMN-Elementen auf einer als Bild vorliegenden Skizze (siehe Abbildung 5.1). Dieser wurde entwickelt, um den natürlichen Offline-Datensatz hdBPMN [27] zu annotieren. Das Softwarewerkzeug basiert dabei auf dem Open-Source-Modellierer *bpmn-js* [36], welcher bereits das digitale Modellieren von BPMN-Diagrammen unterstützt.

Der gesamte Arbeitsprozess des BPMN-Image-Annotators lässt sich in drei verschiedene Teilschritte trennen. Der erste umfasst dabei das Hochladen und das anschließende Darstellen einer auf einem Bild dargestellten BPMN-Skizze. Durch diesen Vorgang lässt sich in einem zweiten Schritt das zugrundeliegende Ausgangsbild mittels bpmn-js digital modellieren und rekonstruieren. Hierfür können BPMN-Knoten auf das Ausgangsbild gesetzt und mit passenden BPMN-Kanten verbunden werden. Zusätzlich ist es möglich, Knoten und Kanten mit jeweils einer Textbeschriftung zu versehen. Für den Anwendungsbereich des BPMN-Image-Annotators wurde der Web-Modellierer dabei angepasst. Um eine möglichst granulare Annotation zu ermöglichen, wird beispielsweise das Vergrößern oder Verkleinern aller BPMN-Elemente erlaubt. Des Weiteren werden einige syntaktische Verbindungsregeln ignoriert, um das Annotieren Modellierungsfehler enthaltener Skizzen zu ermöglichen. Der letzte Schritt besteht aus dem Herunterladen einer BPMN-2.0-XML-Datei, welche das modellierte BPMN-Diagramm repräsentiert. Diese XML-Daten enthalten dabei Informationen über alle Bounding-Boxen und Klassifikationen relevanter Knoten und Kanten. Letztere besitzen zusätzlich Referenzen auf ihre zugehörigen Ausgangs- und Ziel-

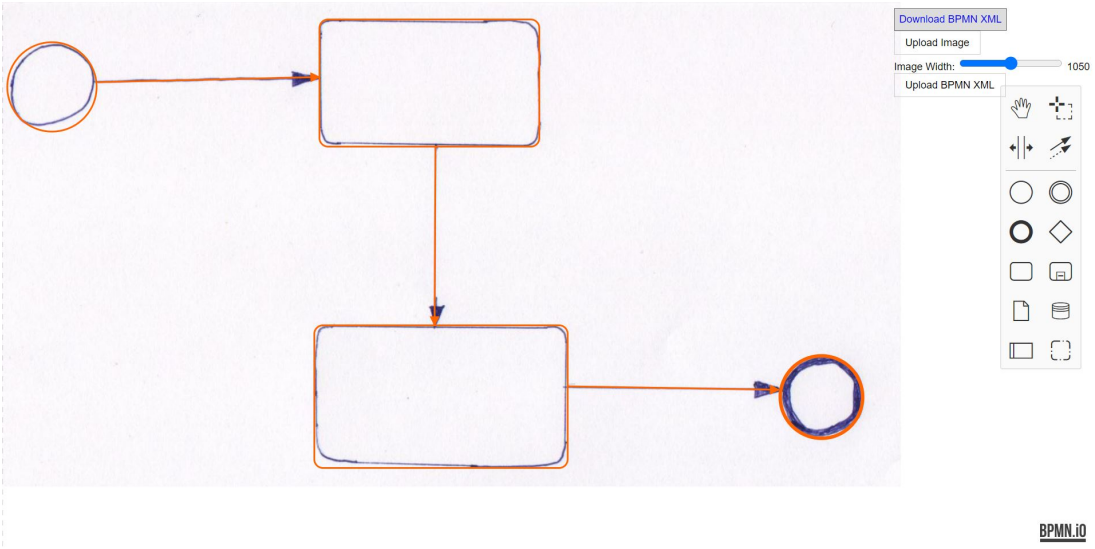


Abbildung 5.1.: Annotation eines handgezeichneten BPMN-Diagramms mit dem BPMN-Image-Annotator. Hierbei wird die Skizze mittels bpmn-js digital auf dem hochgeladenen Bild rekonstruiert. Über die sich rechts befindende Palette können BPMN-Elemente auf das Ausgangsbild platziert werden.

knoten, sowie Felder mit ihren jeweiligen Weg- und Schlüsselpunkten. Jedes Textelement wird außerdem mit Verweis auf das von ihm beschriebene BPMN-Element gesichert.

Im Bereich der UML-Klassendiagramme ist ein solches Werkzeug ebenso wünschenswert, da es eine präzise Annotation von Diagrammskizzen gewährleistet. Der im Rahmen dieser Bachelorarbeit entstandene UML-Image-Annotator erweitert daher die genannten Teilschritte des BPMN-Image-Annotators auf die Domäne der UML-Klassendiagramme. Das Hochladen einer auf einem Bild gespeicherten Diagrammskizze kann hierbei unverändert weiterverwendet werden, da die zugrundeliegenden Bildformate der Skizzen nicht domänenspezifisch sind. Die Haupterweiterung des UML-Image-Annotators liegt deshalb im zweiten und dritten der dargestellten Teilschritte.

Das Modellieren von UML-Klassendiagrammen mittels bpmn-js ist ohne Erweiterung nicht möglich. Der Web-Modellierer unterstützt jedoch die Integration benutzerdefinierter Elemente und Regeln. Zu diesem Zweck lassen sich in bpmn-js enthaltene Module erweitern oder ersetzen, um so die Funktionalität eng auf eine neue Anwendung zuschneiden zu können. Das XML-Ausgabeformat kann dabei ebenfalls auf neue Elemente und Eigenarten einer benutzerdefinierten Diagrammumgebung angeglichen werden. Der im Rahmen dieser Arbeit entwickelte UML-Image-Annotator setzt hier an und erweitert bpmn-js um UML-Klassendiagrammelemente.

Um eine Verarbeitung dieser zu ermöglichen, werden diese zunächst in einer JSON-Datei spezifiziert. Die benutzerdefinierten UML-Elemente erweitern dabei bestehende BPMN-Elemente, welche jeweils ein ähnliches Modellierungsverhalten vorzuweisen haben. Eine Erweiterung bereits in bpmn-js existierender Elemente ermöglicht es, implementierte und bestehende Funktionalitäten weiterzuverwenden. Dies betritt insbesondere Funktionen, welche unabhängig von der Diagrammart für eine Modellierung dieser sinnvoll sind.

Inbegriffen sind hier unter anderem das Bewegen von Elementen, das automatisierte Speichern aller Bounding-Boxen im Ausgabeformat oder das Verbinden zweier Knoten mittels einer Kante. Des Weiteren werden im JSON-Format domänenspezifische und zusätzliche Eigenschaften neuer Elemente definiert, welche in dieser Form im Bereich der BPMN-Diagramme nicht existieren. Ein Beispiel hierfür ist die Klassifikation von Textblöcken. In der BPMN [28] kann jedes Element maximal einen zugehörigen Textblock enthalten. Eine Separation verschiedener Textarten findet aus diesem Grund nicht statt. Bei UML-Klassendiagrammen existieren jedoch verschiedenste Typen von Text (siehe Unterabschnitt 2.1.1). Klassen können beispielsweise sowohl einen Namen als auch mehrere Attribute oder Methoden besitzen. Um zwischen diesen Textblöcken bei der Annotation zu unterscheiden, ist eine Definition dieser essenziell (siehe Abbildung 5.2).

```
{
    "name": "Label",
    "superClass": [
        "bpmn:TextAnnotation"
    ],
    "properties": [
        ...,
        {
            "name": "label_type",
            "type": "LabelType",
            "isAttr": true
        }
    ]
},
{
    "name": "LabelType",
    "literalValues": [
        {
            "name": "method"
        },
        {
            "name": "attribute"
        },
        ...
    ]
}
```

Abbildung 5.2.: Definition eines benutzerdefinierten UML-Textblocks *„Label“*, welcher das bereits in bpmn-js existierende Element *„bpmn:TextAnnotation“* erweitert. Das neudefinierte Attribut *„label_type“* speichert dessen zugehörige UML-Textart.

Durch eine solche Definition aller zu annotierenden UML-Elemente ist eine standardmäßige Verarbeitung durch eine Vielzahl in bpmn-js enthaltener Module bereits möglich.

Diese können so ohne Weiteres in einer neuen Diagrammdomäne wiederverwendet werden. Für das Modellieren mit UML-Klassendiagrammelementen in bpmn-js existieren zusätzlich weitere domänenspezifische Funktionalitäten. Der Hauptbestandteil des UML-Image-Annotators ist dabei ein benutzerdefinierter Modellierer, welcher den Standard-BPMN-Modellierer für den neuen Anwendungsbereich ersetzt. Dieser wird durch die Klasse *UmlModeler* repräsentiert und beinhaltet verschiedene UML-Module (siehe Abbildung 5.3), welche speziell auf die Domäne der UML-Klassendiagramme zugeschnitten sind. Diese Module werden im Folgenden mitsamt der daraus resultierenden Gesamtfunktionalität präsentiert.

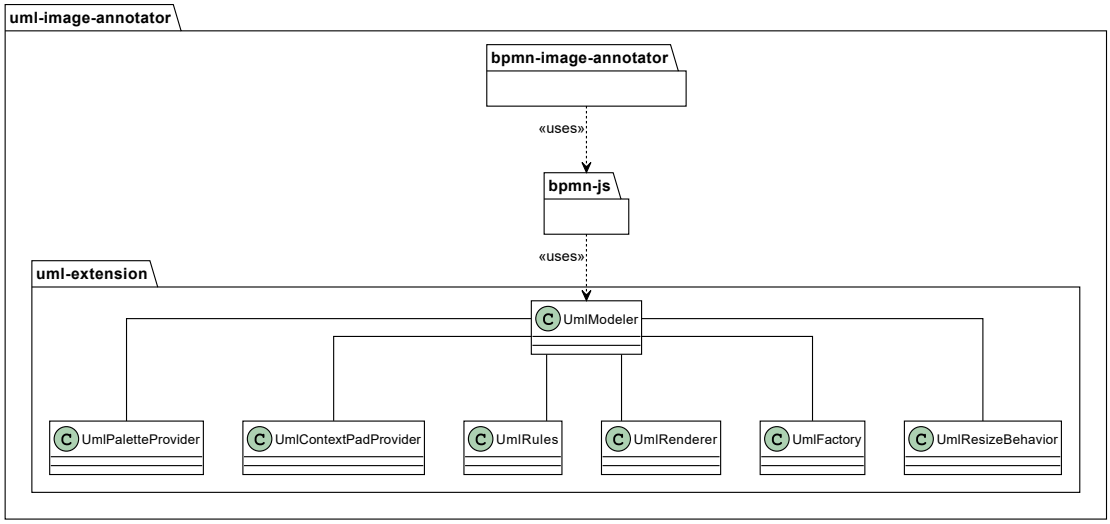


Abbildung 5.3.: Systemarchitektur des UML-Image-Annotators. Dieser erweitert bpmn-js um den benutzerdefinierten Modellierer *UmlModeler*.

UmlPaletteProvider Um die auf einer hochgeladenen Diagrammskizze dargestellten Knoten mittels bpmn-js rekonstruieren zu können, bedarf es ein Mittel, diese zu erzeugen. Normale BPMN-Elemente können zu diesem Zweck standardmäßig über die sogenannte *Palette* ausgewählt und auf das zugehörige Bild gesetzt werden. Für die Übertragung auf eine neue Diagrammdomäne müssen die auf dieser Palette dargestellten Knoten folglich durch benutzerdefinierte UML-Elemente ersetzt werden (siehe Abbildung 5.4). Das Modul *UmlPaletteProvider* definiert zu diesem Zweck alle Einträge, welche identisch mit denen in Tabelle 4.2 identifizierten Knotentypen sind.

UmlRenderer Wird ein Knoten über die Palette ausgewählt, so muss definiert sein, in welcher Form dieses Element auf das zugrundeliegende Ausgangsbild gezeichnet wird. Diesem Zweck dient der *UmlRenderer*. Dieser ordnet jedem benutzerdefinierten Knoten und jeder Kante eine eindeutige, geometrische Form zu und zeichnet sie nach Auswahl auf das Ausgangsbild. Klassen werden so beispielsweise durch ein rotes Rechteck repräsentiert, während Schnittstellen durch ein grünes Rechteck symbolisiert werden. Die initialen

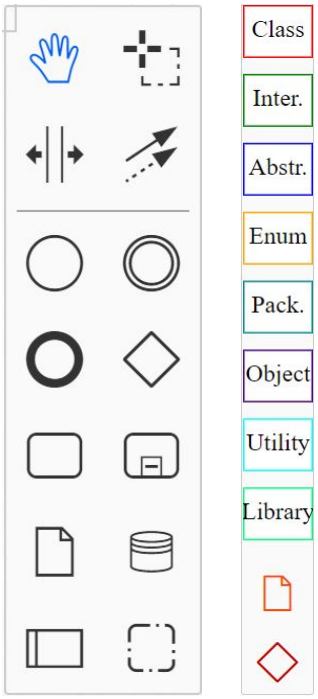


Abbildung 5.4.: **(a)** Standard-BPMN-Palette des BPMN-Image-Annotators und **(b)** benutzerdefinierte Palette des UML-Image-Annotators

Startgrößen dieser geometrischen Formen sind dabei nach Erstellung dieser frei verstellbar, um so eine möglichst präzise Annotation der zugehörigen Bounding-Boxen zu ermöglichen.

UmlContextPadProvider Über die Palette wird bislang lediglich die Erzeugung von Knoten ermöglicht. Das Verbinden dieser mittels verschiedenster UML-Kantenarten oder das Erzeugen von Textblöcken ist so noch nicht darstellbar. Hierfür kann unter anderem das *Context-Pad* genutzt werden. Dieses wird über das Anwählen eines bereits existierenden Elements aufgerufen und beinhaltet für dieses spezifisch ausführbare Aktionen. Die Aktionen werden für jedes definierte UML-Element in der Klasse *UmlContextPadProvider* festgelegt. Die Context-Pad-Einträge eines Knotens umfassen dabei unter anderem das Verbinden mittels spezifischer Kantenarten oder das auf den Knotentyp zugeschnittene Erstellen von Textblöcken (siehe Abbildung 5.5). Auf diese Weise ist es möglich, erstellte Textblöcke und Kanten direkt bei ihrer Instanziierung mit Referenz auf ihren zugehörigen Knoten zu sichern. Das Erstellen eines kantenbeschreibenden Textblocks ist analog über das Context-Pad dieser Kante ausführbar (siehe Abbildung 5.6). Bei Assoziationen, Aggregationen und Kompositionen ist es zudem zusätzlich möglich, eine Pfeilspitze hinzuzufügen oder zu entfernen. Diese Änderung äußert sich sowohl visuell während der Annotation, als auch an dem zusätzlich definierten Attribut „*has_arrowhead*“ im XML-Ausgabeformat. Dies ermöglicht eine spätere Separation von Kanten mit und ohne Pfeilspitzen bei der Anwendung eines maschinellen Lernverfahrens. Des Weiteren ist aus gleichem Grund

eine Trennung der Assoziation in zwei separate Kategorien innerhalb von bpmn-js nicht notwendig.

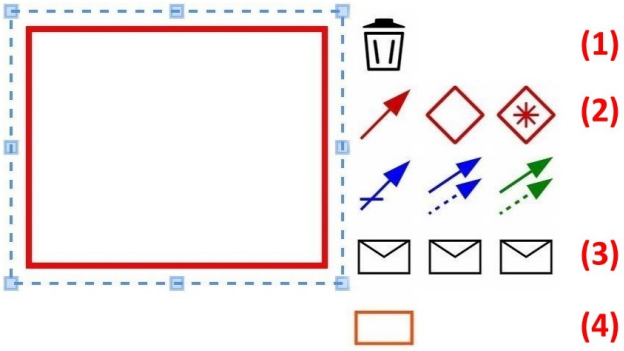


Abbildung 5.5.: Benutzerdefiniertes Context-Pad einer UML-Klasse. Ermöglicht **(1)** das Entfernen der Klasse, **(2)** das Verbinden mittels verschiedener Kantentypen, **(3)** das Hinzufügen unterschiedlicher Textarten, sowie **(4)** das Anfügen eines Qualifizierers.

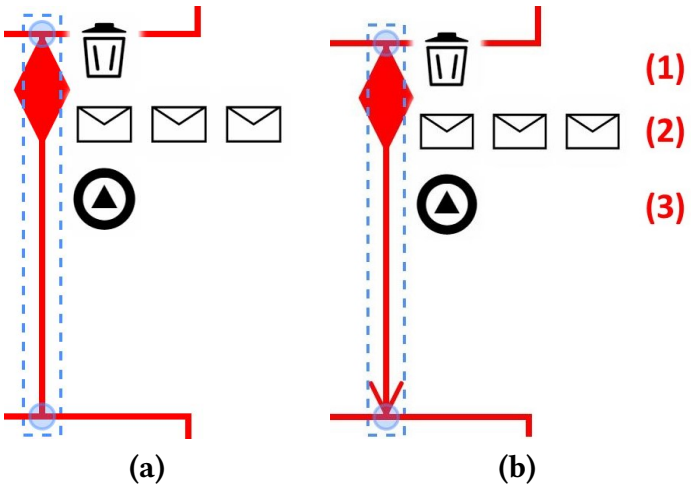


Abbildung 5.6.: Benutzerdefiniertes Context-Pad einer Komposition **(a)** ohne und **(b)** mit Pfeilspitze. Ermöglicht **(1)** das Entfernen der Komposition, **(2)** das Hinzufügen unterschiedlicher Textarten, sowie **(3)** das Hinzufügen oder Entfernen einer Pfeilspitze.

UmlRules In bpmn-js werden standardmäßig syntaktische Regeln der BPMN-Domäne umgesetzt. Hieraus lässt sich ableiten, wann ein Element gezeichnet und mit welcher Kantenart es verbunden werden darf. Eine analoge Definition ist auch für eine Erweiterung der UML-Klassendiagramme notwendig. Die Klasse *UmlRules* definiert daher das erlaubte Verhalten von UML-Elementen. Hierbei ist vor allem wichtig, dass diese ohne Ausnahme

auf dem zugrundeliegenden Ausgangsbild platziert und vergrößert oder verkleinert werden können. Des Weiteren muss das Ablegen von Textblöcken auf bereits gesetzte Knoten und Kanten erlaubt werden, um beispielsweise das Annotieren der Bounding-Boxen von Textblöcken innerhalb eines Knotens zu ermöglichen. Zusätzlich müssen Verbindungsaktionen vom Context-Pad wahrgenommen und jeweils mit dem ausgewählten Kantentyp durchgeführt werden. Eine Definition syntaktischer UML-Verbindungsregeln ist innerhalb dieser Klasse ebenfalls möglich. Wie bereits in Kapitel 4 dargestellt, ist jedoch das Annotieren fehlerhafter Daten für das maschinelle Lernen durchaus angedacht. Aus diesem Grund werden für das Annotieren jegliche Verbindungsregeln ignoriert. Ähnlich wurde auch beim BPMN-Image-Annotator verfahren, der ebenfalls einige syntaktische BPMN-Regeln vernachlässigt [15].

Weitere Module Weiterhin existieren innerhalb der UML-Erweiterung die Klassen *UmlFactory* und *UmlResizeBehavior*. Erstere definiert dabei Besonderheiten hinsichtlich des Speicherns von UML-Klassendiagrammelementen innerhalb des BPMN-2.0-XML-Formats. Die automatisierte Speicherung der Annotationen wird bereits durch die Spezifikation der zuvor genannten JSON-Datei weitestgehend gewährleistet. Innerhalb der UmlFactory werden daher nur kleinere Änderungen im Speichern von Elementidentifikatoren spezifiziert, um diese weiter auf die neue Diagrammdomäne zuzuschneiden. Die Klasse UmlResizeBehavior definiert hingegen zusätzliche Regeln hinsichtlich der Mindestgröße von Textblöcken, welche zuvor an die jeweilige Superklasse ausgerichtet war (siehe Abbildung 5.2). Diese wird auf ein Minimum reduziert, da vor allem Multiplizitäten oft nur aus einem Textzeichen bestehen und ihre Bounding-Box dementsprechend klein ist. Diese Änderung ermöglicht somit ein möglichst granules Annotieren von Text.

Resultierende Gesamtfunktionalität Insgesamt ermöglichen die genannten Klassen, in Zusammenarbeit mit bestehenden Modulen von bpmn-js, das präzise Annotieren von UML-Klassendiagrammskizzen. Nach dem Hochladen einer auf einem Bild dargestellten Skizze, lassen sich mit der präsentierten Erweiterung benutzerdefinierte UML-Elemente digital über dem Ausgangsbild rekonstruieren. Diese werden weiterhin automatisiert in einem XML-Format mit zugehöriger Bounding-Box und Kategorie gespeichert. Kanten enthalten zudem, analog zum BPMN-Image-Annotator, Informationen über ihre jeweiligen Schlüssel- und Wegpunkte, sowie Referenzen auf deren Ausgangs- und Zielknoten. Die Annotation von Textblöcken ist zudem noch präziser und granularer möglich als beim ursprünglichen BPMN-Image-Annotator. Diese werden aufgrund der bereits genannten Existenz verschiedener Textarten, intern auf eine andere Weise verarbeitet als standardmäßiger BPMN-Text. Um Text überall platzieren und auch innerhalb eines Elements zwischen unterschiedlichen Typen separieren zu können, werden diese als eine Art Knoten definiert. Auf diese Weise lässt sich die Position und Größe der Bounding-Box frei verändern. Beim BPMN-Image-Annotator ist dies im Falle von knotenbeschreibenden Text nicht möglich. Die im Ausgabeformat gespeicherte Bounding-Box von Text richtet sich hier ausschließlich an die digitale Textgröße und -länge. Die handgeschriebene Schrift der Ausgangsskizze kann dabei nicht berücksichtigt werden. Dies muss beim Annotieren der BPMN-Elemente in einem separaten Schritt nachbearbeitet werden. Der UML-Image-Annotator erweitert

somit nicht nur den BPMN-Image-Annotator erfolgreich auf eine neue Diagrammdomäne, sondern verbessert und verfeinert ihn in einigen Punkten.

5.2. Annotieren von UML-Klassendiagrammskizzen

Die in Kapitel 4 präsentierten 115 Klassendiagrammskizzen wurden nach deren Erhebung mit Hilfe des UML-Image-Annotators manuell annotiert. Hierbei wurden alle enthaltenen UML-Elemente (siehe Tabelle A.3) mitsamt ihrer Bounding-Box und Kategorie präzise markiert. Das Annotieren von Kanten stellt dabei eine Besonderheit dar. Wie Abbildung 2.6 und verwandte Arbeiten [15, 16] verdeutlichen, werden Diagrammskizzen teilweise sehr unpräzise gezeichnet. Dies äußert sich unter anderem an Ausgangs- und Zielpunkten von Kanten, welche nicht direkt an Knoten haften. Eine Annotation dieser ungenau gezeichneten Schlüsselpunkte könnte folglich zu einer Erkennung von Kanten ohne zugehörige Ausgangs- oder Zielknoten führen. Aus diesem Grund wurden stattdessen genau die Punkte markiert, an denen eine Kante bei einer korrekten und präzisen Darstellung die zugehörigen Knoten geschnitten hätte (siehe Abbildung 5.7). Ein maschinelles Lernverfahren soll so nach Training des Datensatzes in der Lage sein, auch für nicht präzise gezeichnete Kanten einen Quell- und Zielkonten vorherzusagen. Auf diese Weise können Diagrammzusammenhänge und die daraus resultierende Gesamtstruktur erkannt werden. Das dargestellte Vorgehen der Annotation orientiert sich an der Arbeit von Schäfer u. a. [15]. In dieser wurde am Datensatz hdBPMN gezeigt, dass eine solche Annotation von Pfeilschlüsselpunkten die Erkennung dieser verbessert.

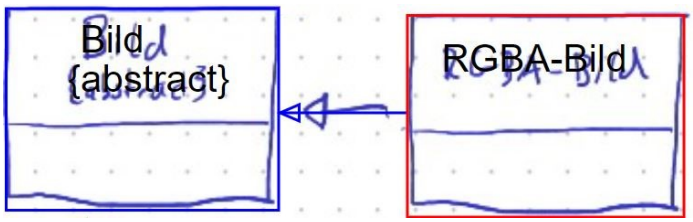


Abbildung 5.7.: Annotation des Ausschnitts aus Abbildung 2.6. Statt ungenau gezeichneter Kantenenden werden die Stellen markiert, an denen die Kante bei einer präziseren Darstellung tatsächlich hätte enden sollen.

Für jeden Textblock innerhalb der handgezeichneten Klassendiagramme wurde zudem, neben der Bounding-Box und Kategorie, der zugehörige Textinhalt annotiert. Dies macht den Datensatz zusätzlich für eine Evaluation maschineller Lernverfahren mit handschriftlicher Texterkennung einsetzbar. Bei der Trennung von Textblöcken wurde dabei stets die Separation der in Unterabschnitt 2.1.1 eingeführten Textkategorien berücksichtigt.

Die in Kapitel 4 dargestellten syntaktischen Verbindungsfehler, welche unzulässige Kantenarten zwischen zwei Knoten modellieren, wurden allesamt annotiert. Diese behindern maschinelle Lernverfahren im Bereich der Skizzenerkennung nicht, wenn diese in erster Linie die geometrische Elementform für die Erkennung nutzen. Wird die Elementgestalt korrekt dargestellt, so stellt dies folglich kein Problem für das maschinelle Lernen dar. Das Ignorieren Modellierungsfehler enthaltener Daten ist durch die im Zuge der Datenanalyse

entstandene Fehlerdokumentation dennoch weiterhin möglich. Dies kann beispielsweise sinnvoll sein, wenn das zugrundeliegende Verfahren zusätzlich syntaktische Regeln für eine präzisere Klassifikation der erkannten Elemente verwendet. Durchgestrichene Knoten, Kanten oder Textblöcke wie in Abbildung 2.6 wurden hingegen von der Annotation ausgeschlossen. Dies soll maschinelle Lernverfahren nach dem Trainieren des Datensatzes dazu verleiten, diese bei der Erkennung ebenfalls nicht zu berücksichtigen.

Nach der manuellen Annotation aller Skizzen konnten folglich 115 BPMN-2.0-XML-Dateien gesichert werden, welche alle die zugrundeliegenden UML-Diagramme präzise beschreiben. Diese wurden analog zu ihren zugehörigen Bilddateien benannt, um so das Referenzieren in beide Richtungen zu ermöglichen. Des Weiteren besitzt jede XML-Datei analog zum hdBPMN-Datensatz einen Kommentar [15], der auf die bei der Annotation gegebene Bildauflösung verweist. Dadurch ist es auch im Nachhinein möglich, alle spezifizierten Bounding-Boxen passend auf die im Ausgangsbild dargestellten Elemente zu setzen.

5.3. Konvertieren in einen COCO-Datensatz

Wie bereits in Abschnitt 2.3 illustriert, ist das COCO-Datenformat im Bereich der Skizzenerkennung ein passendes Eingabeformat für maschinelle Lernverfahren. Der hdBPMN-Datensatz liegt aus diesem Grund ebenfalls als COCO-Datensatz vor [27]. Für die Transformation der BPMN-Skizzen wurde hierfür das Python-Skript pybpmn [34] eingesetzt. Dieses ist in der Lage, Bilddateien und ihre zugehörigen Annotationen in einen COCO-Datensatz zu konvertieren. Da die Annotationen der UML-Klassendiagramme ebenfalls im BPMN-2.0-XML-Format vorliegen, wird pybpmn für den Zweck dieser Arbeit auf den Kontext der UML-Klassendiagramme angepasst. Im Folgenden wird daher in einem ersten Schritt die Funktionsweise der Ursprungsversion illustriert. Hierbei wird ebenfalls dargestellt, wie die Skizzen des hdBPMN-Datensatzes für den Einsatz eines maschinellen Lernverfahrens separiert wurden. Anschließend werden die am Skript vorgenommenen Änderungen in der angepassten Version *pybpmn-uml* [37] aufgezeigt. Des Weiteren wird der daraus resultierende Aufbau des COCO-Datensatzes dieser Arbeit erläutert.

Skript für das Konvertieren von BPMN-Skizzen Das Ausgangsskript pybpmn erhält als Eingabe sowohl Verweise auf die Bild- als auch auf die XML-Dateien der BPMN-Skizzen. Zusätzlich wird in einer eingegebenen Tabelle definiert, welche der Diagramme jeweils dem *Training*, der *Validierung* und dem *Test* dienen. Für jeden der drei genannten Bereiche wird von pybpmn ein separater COCO-Satz in Form einer JSON-Datei generiert. Der Trainingssatz dient dabei ausschließlich dem Trainieren eines maschinellen Lernverfahrens. Das trainierte Verfahren wird im Anschluss auf dem Validierungssatz validiert, um das Modell des Verfahrens zu verfeinern. Das finale Modell evaluiert schließlich die Erkennung einzelner Diagrammelemente auf dem Testsatz. Der hdBPMN-Datensatz und pybpmn trennen die Skizzen nach Zeichnern, das heißt alle Skizzierenden sind zwischen den generierten Sätzen disjunkt. Dies macht Erkennungsergebnisse aussagekräftiger, da maschinelle Lernverfahren so auf unabhängigen Daten trainiert und geprüft werden.

Weiterhin ist das Skript stark auf den BPMN-Datensatz zugeschnitten. Dies äußert sich folglich an den definierten COCO-Kategorien (analog zu Abbildung 2.11). Diese umfassen genau die BPMN-Elemente, welche Teil des Datensatzes sind. Des Weiteren ist es mit pybpmn möglich, mehrere COCO-Kategorien zusammenzufassen. Ein solcher Vorgang kann bei großen Ähnlichkeiten oder Unterrepräsentationen bestimmter Elementarten sinnvoll sein. Nach Ausführung des Skripts werden drei COCO-Sätze erzeugt, welche jeweils aus dem in Abschnitt 2.3 beschriebenen Bildblock, Annotationsblock und Kategorieblock bestehen.

Skript für das Konvertieren von UML-Klassendiagrammskizzen Um eine Anwendung des pybpmn-Skripts für die annotierten UML-Daten dieser Arbeit zu ermöglichen, sind an einigen Stellen Anpassungen notwendig. Die modifizierte Variante pybpmn-uml setzt so auf eine alternative Art der beschriebenen Datentrennung. Wie in Kapitel 4 illustriert, hat jeder Zeichner innerhalb der zwei Datenquellen genau ein Diagramm angefertigt. Eine analoge Trennung zum hdBPMN-Datensatz ist daher nicht notwendig. Die angepasste Version setzt daher auf eine reine Trennung nach Dateiname. Dies reduziert die Komplexität der Separation und macht das Skript für alternative Trennungsansätze flexibler.

Die zwischen den UML-Daten bestehende Abhängigkeit äußert sich lediglich an jeweils gleichen Aufgabenstellungen der Datenquellen. Um den Einfluss dieser Abhängigkeit auf spätere Erkennungsergebnisse zu reduzieren, werden ausschließlich Skizzen der Studierendenabgabe dem Trainingssatz zugeteilt. Sowohl die übrigen Diagrammdaten dieser Quelle, als auch die der Nutzerstudie, werden schließlich auf den Validierungs- und Testsatz aufgeteilt. Tabelle 5.1 zeigt dabei die Diagrammverteilung auf die jeweiligen Sätze. Tabelle 5.2 präsentiert zusätzlich die daraus resultierende Verteilung aller vorkommenden Knoten, Kanten und Textblöcke.

Trennung	Studierendenabgabe	Nutzerstudie	Gesamtanzahl
Training	77	0	77
Validierung	10	9	19
Test	10	9	19
Gesamt	97	18	115

Tabelle 5.1.: Diagrammverteilung: Training, Validierung und Test

Trennung	Knoten	Kanten	Textblöcke
Training	1119	1215	3916
Validierung	221	236	707
Test	213	226	712
Gesamt	1553	1677	5335

Tabelle 5.2.: Elementverteilung: Training, Validierung und Test (detailliertere Verteilung siehe Tabelle A.4)

Des Weiteren werden alle als COCO-Kategorien definierten BPMN-Elemente durch die im Datensatz enthaltenen Knoten- und Kantentypen ersetzt. Diese werden durch eine einzelne Textkategorie ergänzt, welche weiterhin durch das in bpmn-js zusätzlich definierte Attribut *„label_type“* separiert wird (siehe Abbildung 2.10). Für die Erstellung dieser Kategorien existieren zudem zwei optionale Einstellungen in pybpmn-uml. Die erste dieser umfasst das Zusammenfassen der Kategorien *Klasse*, *abstrakte Klasse*, *Schnittstelle*, *Enumeration*, *Objekt*, *Utility* und *Bibliothek*. Zwischen diesen Elementen existieren keine Unterschiede hinsichtlich ihrer geometrischen Gestalt. Die Separation erfolgt lediglich in Form von Text (siehe Unterabschnitt 2.1.1). Maschinelle Lernverfahren ohne Texterkennung sind somit nicht in der Lage, zwischen diesen zu unterscheiden. Des Weiteren treten einige der genannten Kategorien im Vergleich zu anderen stark unterrepräsentiert im Datensatz auf (siehe Tabelle A.3). Eine Vereinigung dieser zu einer gemeinsamen Kategorie mit dem Namen *Klassenknoten* kann daher aus den genannten Gründen durchaus sinnvoll sein. Die zweite Einstellung ermöglicht das Separieren der Assoziationskategorie in bidirektional und unidirektional. Wie in Abschnitt 5.1 dargestellt, trennt der UML-Image-Annotator diese durch ein zusätzlich definiertes Attribut *„has_arrowhead“*. Dieses XML-Attribut kann pybpmn-uml nutzen und die Assoziationsobjekte in zwei verschiedene Kategorien trennen. Ein solcher Vorgang kann sinnvoll sein, wenn das zugrundeliegende maschinelle Lernverfahren explizit eine dieser Unterkategorien evaluieren möchte. Des Weiteren erstellt pybpmn-uml für jede Kante, auf Grundlage der Kategorie und dem Pfeilspitzenstatus, ein zusätzliches Attribut *„directed“*. Dies ermöglicht maschinellen Lernverfahren ohne UML-Domänenwissen das sofortige Trennen von gerichteten und ungerichteten Kanten.

Insgesamt können durch die dargestellten Modifikationen mehrere COCO-Varianten des in dieser Arbeit präsentierten UML-Datensatzes generiert werden. Tabelle 5.3 zeigt dabei die Unterschiede dieser Versionen auf.

Version	Vereinigung der Klassenknoten	Separation der Assoziation
1	-	-
2	-	✓
3	✓	-
4	✓	✓

Tabelle 5.3.: COCO-Varianten des Datensatzes

6. Evaluation

Die Evaluation des Datensatzes lässt sich in zwei Teile trennen. Abschnitt 6.1 bewertet zunächst den Datensatz anhand seiner Vielfältigkeit, Detailtiefe und Größe. In diesem Schritt wird dieser außerdem direkt mit dem natürlichen Offline-Datensatz hdBPMN verglichen. Der zweite Teil der Evaluation bewertet einen expliziten Einsatz des Datensatzes für ein maschinelles Lernverfahren. Dieses basiert auf einem angepassten DiagramNet-Ansatz, welcher in Abschnitt 6.2 dargestellt wird. Die dabei entstandenen Ergebnisse werden anschließend interpretiert und eingeordnet, um zu evaluieren, inwiefern sich der Datensatz für den Einsatz maschineller Lernverfahren eignet.

6.1. Eigenschaften des Datensatzes

Der im Rahmen dieser Bachelorarbeit entstandene Datensatz umfasst 115 handgezeichnete UML-Klassendiagramme. Die enthaltenen Skizzen wurden dabei innerhalb einer Datenquelle von jeweils verschiedenen Zeichnern ohne Vorlage angefertigt. Die Skizzierenden waren dadurch ausschließlich an die gegebene Aufgabenstellung und nicht an weitere Strukturen gebunden, wie es beispielsweise bei bestehenden Online-Datensätzen von Flussdiagrammen [32] der Fall gewesen ist. Wie in Kapitel 4 bereits illustriert wurde, resultiert daraus eine Varianz der Daten bezüglich verschiedenster Eigenschaften händisch angefertigter Diagramme. Die gegebene Vielfältigkeit und das Existieren für Skizzen typischer Ungenauigkeiten machen die Daten somit möglichst realistisch.

Die darauf aufbauenden Annotationen beinhalten exakte Bounding-Boxen aller der sich in den Skizzen befindenden Knoten, Kanten und Textblöcke. Diese wurden allesamt mit einer passenden UML-Kategorie klassifiziert. Zusätzlich wurden Weg- und Schlüsselpunkte aller Kanten markiert. Alle getätigten Textannotationen bestehen weiterhin aus einer Referenz auf das von ihm beschriebene Element, dem genauen Textinhalt und einer zugehörigen UML-Textkategorie. Die Annotationen des Datensatzes sind folglich präzise und vollständig. Die Verfügbarkeit dieser im COCO-Datenformat erlaubt zudem den sofortigen Einsatz für maschinelle Lernverfahren wie Arrow R-CNN [14] oder das DiagramNet-Verfahren [16].

Ein vergleichbarer, annotierter Datensatz von UML-Klassendiagrammskizzen ist zum jetzigen Zeitpunkt nicht veröffentlicht. Mit dem hdBPMN-Datensatz [27] existiert nach aktuellem Kenntnisstand zudem nur ein weiterer natürlicher Offline-Datensatz einer vergleichbaren Diagrammdomäne. Dieser verfügt dabei mit 502 BPMN-Diagrammen über weitaus mehr Skizzen als der Datensatz dieser Arbeit. Dies äußert sich vor allem an einer größeren Anzahl von Knoten- und Kantenelementen, welche für das Training, das Validieren und das Testen eines maschinellen Lernverfahrens eingesetzt werden können. Die Anzahl der in den Skizzen enthaltenen Textblöcke ist mit 3040 [15] jedoch gerin-

ger als die des vorgestellten UML-Datensatzes mit 5335 (siehe Tabelle 5.2). Begründbar ist dies durch die in der UML existierende Vielfalt verschiedener Textkategorien (siehe Unterabschnitt 2.1.1). Ein Einsatz des Datensatzes für maschinelle Lernverfahren mit handschriftlichem Texterkennungssystem ist daher ebenfalls denkbar.

Insgesamt lässt sich aus den genannten Punkten folgern, dass der Datensatz dieser Arbeit dem in Kapitel 1 dargestellten Problem, der Nichtverfügbarkeit eines annotierten Datensatzes von UML-Klassendiagrammskizzen, entgegenwirkt. Die Eigenschaften der Diagramme und die Annotationen sind dabei dem für das maschinelle Lernen bereits erprobten Offline-Datensatz hdBPMN sehr ähnlich. Durch die in der Arbeit präsentierten Werkzeuge zur Annotation und Konvertierung von UML-Klassendiagrammen wird zudem das Erweitern des Datensatzes erleichtert. Eine solche Erweiterung kann zukünftig die Anzahl der im Datensatz enthaltenen Diagramme und folglich auch die Häufigkeiten der bereits auftretenden Knoten- und Kantenkategorien weiter erhöhen.

6.2. Einsatz für ein maschinelles Lernverfahren

Um den Datensatz auf die Erkennung handgezeichneter UML-Klassendiagramme zu testen, wird dieser für das maschinelle Lernverfahren DiagramNet [16] von Schäfer und Stuckenschmidt eingesetzt. Dieses wurde von den Entwicklern des Verfahrens für den Kontext der Bachelorarbeit vereinfacht. Das angewandte Lernverfahren wird dabei in Unterabschnitt 6.2.1 vorgestellt. Die auf Basis des UML-Datensatzes resultierenden Erkennungsergebnisse werden anschließend in Unterabschnitt 6.2.2 präsentiert.

6.2.1. DiagramNet

Verfahren Der für diese Arbeit vereinfachte DiagramNet-Ansatz [16] setzt sich aus drei Stufen zusammen. In einem ersten Schritt erkennt das Verfahren mittels Faster R-CNN [25] eine Knotenmenge V. Für jeden Knoten $v \in V$ wird dabei eine zugehörige Bounding-Box $b_v \in \mathbb{R}^4$ und eine UML-Kategorie c_v vorhergesagt. Die Vorhersage wird zusätzlich mit einer Wahrscheinlichkeit p_v belegt. Beträgt p_v für einen Knoten $v \in V$ weniger als 70%, so wird v aus der Knotenmenge V entfernt. Des Weiteren werden in diesem Schritt potenzielle Duplikate durch die sogenannte *Intersection over Union* (IoU) aussortiert. Die IoU quantifiziert dabei die Überlappung zweier Bounding-Boxen. Überschreitet diese 80% bei zwei Knoten $v_1, v_2 \in V$, so wird das Element mit der geringeren Wahrscheinlichkeit aus der Menge V entfernt.

In einem nächsten Schritt generiert das Verfahren anschließend potenzielle Kantenkandidaten. Dazu werden jeweils die Knotenpaare $(v_1, v_2) \in E \subseteq V \times V$ gebildet. Durch das Anwenden syntaktischer Verbindungsregeln der UML, könnte man bereits zu diesem Zeitpunkt fehlerhafte Paare aussortieren. Modellierungsfehler dieser Art sind jedoch, wie in Kapitel 5 illustriert, regulärer Bestandteil des Datensatzes dieser Arbeit. Das Ausschließen syntaktisch falscher Paare findet somit im angewandten Verfahren nicht statt.

Auf Basis der generierten Kandidaten werden in einem letzten Schritt die endgültigen Kantenvorhersagen getroffen. Für jeden Kantenkandidat $(v_1, v_2) \in E$ wird hierfür als Erstes die Bounding-Box $b_{(v_1,v_2)} \in \mathbb{R}^4$ durch das Vereinigen von $b_{v_1} \in \mathbb{R}^4$ und $b_{v_2} \in \mathbb{R}^4$

vorhergesagt. Anschließend wird die potenzielle Kante mit einer möglichen UML-Kategorie $c_{(v_1,v_2)}$ klassifiziert und zugehörige Wegpunkte prognostiziert. Zusätzlich werden diese Prognosen, analog zu den Knoten, mit einer Wahrscheinlichkeit $p_{(v_1,v_2)}$ belegt. Kanten $(v_1, v_2) \in E$ mit $p_{(v_1,v_2)} < 70\%$ werden als Folge dessen aus der Kantenmenge entfernt. Für jede Kante wird die finale Bounding-Box im Anschluss aus den prognostizierten Wegpunkten erstellt. Diese ist dabei das kleinste minimale Rechteck, welches aller der vorhergesagten Wegpunkte beinhaltet.

Speziell für den Datensatz dieser Arbeit wird das Modell für ungerichtete Assoziationen $\{v_1, v_2\} \in E$ auf die Erkennung beider Richtungen trainiert. Wird dabei sowohl (v_1, v_2) als auch (v_2, v_1) erkannt, so wird das Paar mit der geringeren, vorhergesagten Wahrscheinlichkeit aus der Kantenmenge E entfernt.

Evaluationsmetriken Die vorhergesagten Erkennungen werden auf Basis der in Abschnitt 5.2 beschriebenen Annotation bewertet. Diese wird im Folgenden als *Grundwahrheit* referenziert.

Eine Erkennung eines Knotens ist genau dann korrekt, wenn der IoU zwischen dem vorhergesagten Knoten und einem zugehörigen Element der Grundwahrheit mindestens 80% beträgt. Zusätzlich muss der Knoten mit der gleichen UML-Kategorie klassifiziert worden sein. Eine korrekte Klassifizierung ist weiterhin für die Erkennung von Kanten notwendig. Das Erkennen dieser setzt ebenfalls eine passende Vorhersage der zugehörigen Quell- und Zielknoten voraus. Für die Evaluation des Datensatzes dieser Bachelorarbeit ist zudem das Bewerten ungerichteter Kanten ein Novum. Eine in der Grundwahrheit enthaltene bidirektionale Kante $\{v_1, v_2\}$ wird dabei ebenfalls als korrekt erkannt eingestuft, wenn die Erkennung (v_1, v_2) in gegenläufiger Richtung der Annotation (v_2, v_1) verläuft.

Unter Erfüllung aller der genannten Kriterien kann eine Vorhersage als *richtig positiv* deklariert werden. *Falsch positiv* ist eine Erkennung hingegen, wenn eine solche Zuordnung zu keinem Element der Grundwahrheit gegeben ist. Wenn ein annotiertes Element der Grundwahrheit fälschlicherweise nicht erkannt wurde, so ist die Erkennung *falsch negativ*. Über eine Einordnung dieser Art lassen sich die Mengen aller richtig positiven (TP), falsch positiven (FP) und falsch negativen (FN) Erkennungen bilden. Auf Basis dieser können verschiedene Evaluationsmetriken berechnet werden, welche die Güte der Diagrammerkennung quantifizieren.

Die *Genauigkeit P* definiert dabei den Anteil aller positiven Erkennungen, welche korrekterweise als diese klassifiziert wurden (siehe Gleichung 6.1).

$$P = \frac{|TP|}{|TP| + |FP|} \tag{6.1}$$

Die *Trefferquote R* definiert hingegen den Anteil aller erkannten Elemente der Grundwahrheit (siehe Gleichung 6.2).

$$R = \frac{|TP|}{|TP| + |FN|} \tag{6.2}$$

Über diese Metriken lässt sich das F_1-*Maß* bestimmen (siehe Gleichung 6.3). Dieses berücksichtigt sowohl die Genauigkeit als auch die Trefferquote und quantifiziert somit ein Mittelmaß dieser Werte.

$$F_1 = 2 * \frac{P * R}{P + R} \tag{6.3}$$

6.2.2. Ergebnisse

Der Datensatz wurde für das Training des DiagramNet-Verfahrens in mehrere Versionen des COCO-Datenformats konvertiert (siehe Abschnitt 5.3). Teil der Betrachtung werden im Folgenden insbesondere Version 2 und 4 aus Tabelle 5.3, da beide Versionen die Assoziationskategorie spalten. Auf diese Weise ist eine separate Untersuchung der Erkennung von unidirektionalen und von bidirektionalen Assoziationen möglich. Der Unterschied der Versionen äußert sich in der Anzahl der existierenden Knotenkategorien innerhalb des COCO-Datensatzes.

In einem ersten Schritt werden die Erkennungsergebnisse des DiagramNet-Verfahrens auf Version 2 vorgestellt. Hierbei werden ausschließlich Knoten betrachtet, da sich der Aufbau der beiden Variationen bezüglich Kanten und Text nicht unterscheidet. Anschließend werden die Resultate mit zwei verschiedenen Modellen des Lernverfahrens auf Version 4 präsentiert.

Version 2 Nach Trainieren des zugehörigen Trainingssatzes wurde das vereinfachte DiagramNet-Verfahren auf die Erkennung der Knoten des Validierungssatzes evaluiert. Hierfür wurden für jeden Knoten sowohl Genauigkeit, Trefferquote als auch F_1-Maß bestimmt. Der Validierungssatz enthält dabei nicht jede der im Zuge der Arbeit erhobenen Knotenkategorien. Knotentypen, welche nur in geringem Maße im Datensatz vorkommen, finden sich zu großen Teilen ausschließlich im Trainingssatz wieder (siehe Tabelle A.4). Eine Evaluierung der Erkennung dieser wird daher im Folgenden nicht vorgenommen. Tabelle 6.1 zeigt die entstandenen Ergebnisse auf den im Validierungssatz enthaltenen Knoten.

Kategorie	**N**	**P**	**R**	**F_1**
Klasse	186	94,2	96,8	95,5
Schnittstelle	13	62,5	38,5	47,6
Enumeration	11	72,7	72,7	72,7
Abstrakte Klasse	10	33,3	30,0	31,6
Bibliothek	1	0,0	0,0	0,0

Tabelle 6.1.: Häufigkeit N, Genauigkeit P (in %), Trefferquote R (in %) und F_1-Maß (in %) aller Knoten des Validierungssatzes von Version 2

Die Werte zeigen, dass nach dem Training speziell UML-Klassen akkurat erkannt werden. Bei diesen ist sowohl die Genauigkeit als auch die Trefferquote sehr hoch, was sich folglich am zugrundeliegenden F_1-Maß äußert. Trotz einer geringeren Elementanzahl sind diese Erkennungsraten auf einem ähnlichen Niveau wie von vergleichbaren Knoten des hdBPMN-Datensatzes [16]. Die anderen UML-Kategorien weichen hingegen stark von diesen Ergebnissen ab. Dies ist durch die deutlich niedrigere Häufigkeit im Vergleich zu

den UML-Klassen begründbar. Dennoch zeigen insbesondere die Werte der Enumerationen, dass trotz identischer geometrischer Knotenformen eine Separation bei der automatisierten Erkennung möglich sein kann. Durch das Verfügen größerer Trainings- und Testmengen könnten hier eventuell noch bessere Ergebnisse erzielt werden.

Version 4 Die Elementkategorien aus Tabelle 6.1 unterscheiden sich lediglich in einer Textzeile. Da das DiagramNet-Verfahren nicht über ein handschriftliches Texterkennungssystem verfügt, werden diese in einem zweiten Schritt der Evaluation zu einer einzelnen Kategorie mit dem Namen Klassenknoten vereinigt. Durch diese Vereinigung wird zusätzlich dem Problem geringerer Elementhäufigkeiten einzelner Knotenkategorien entgegengewirkt. Die im Folgenden präsentierten Ergebnisse beruhen dabei auf zwei verschiedenen Modellen. Das erste dieser ist das Standardmodell *M1*, welches analog bei der Validierung von Version 2 eingesetzt wurde. Das zweite der validierten Modelle *M2* wurde im Vorfeld zusätzlich auf dem hdBPMN-Datensatz trainiert. Tabelle 6.2 stellt dabei die Erkennungsergebnisse der beiden Modelle auf dem Validierungssatz dar. Tabelle 6.3 präsentiert analog die Ergebnisse auf dem Testsatz von Version 4.

		Modell M1			**Modell M2**		
Kategorie	**N**	**P**	**R**	**F_1**	**P**	**R**	**F_1**
Klassenknoten	221	99,5	99,5	99,5	100,0	99,5	99,8
Textblock	707	92,4	92,6	92,5	92,9	92,9	92,9
Aggregation	22	80,0	54,5	64,9	81,2	59,1	68,4
Komposition	26	87,0	76,9	81,6	92,0	88,5	90,2
Vererbung	63	77,6	71,4	74,4	84,2	76,2	80,0
Abhängigkeit	17	54,5	35,3	42,9	52,9	52,9	52,9
Realisierung	40	62,9	55,0	58,7	74,3	65,0	69,3
Ger. Assoziation	23	50,0	56,5	53,1	50,0	52,2	51,1
Unger. Assoziation	45	77,4	53,3	63,2	83,9	57,8	68,4
Kanten (gesamt)	236	71,4	60,2	65,3	76,6	66,5	71,2

Tabelle 6.2.: Häufigkeit N, Genauigkeit P (in %), Trefferquote R (in %) und F_1-Maß (in %) aller Elemente des Validierungssatzes von Version 4

Ein direkter Vergleich der beiden Modelle zeigt, dass insbesondere die Erkennung von Kanten mit dem auf dem hdBPMN-Datensatz vortrainierten Modell M2 erkennbar verbessert wurde. Dies äußert sich an einem erhöhten F_1-Maß aller Kanten von 66,5% auf 71,2% beim Validierungssatz und von 70,5% auf 76,9% beim Testsatz. Die Erkennung von Knoten und Textblöcken bleibt dabei auf einem ähnlichen Niveau. Das Transferlernen für die Erkennung von BPMN-Skizzen zu handgezeichneten UML-Klassendiagrammen führt somit insbesondere bei Kanten zu Verbesserungen. Für die Einordnung expliziter Erkennungsergebnisse einzelner Elementkategorien werden folglich die Werte von Modell M2 referenziert.

Diese zeigen, dass Klassenknoten nach der Vereinigung dieser höchst akkurat erkannt werden. Dies äußert sich gleichermaßen an einer sehr hohen Genauigkeit und Trefferquote.

		Modell M1			Modell M2		
Kategorie	**N**	**P**	**R**	**F_1**	**P**	**R**	**F_1**
Klassenknoten	213	99,5	99,5	99,5	98,6	99,5	99,1
Textblock	712	87,6	88,1	87,8	88,7	86,9	87,8
Aggregation	27	100,0	55,6	71,4	94,4	63,0	75,6
Komposition	27	78,1	92,6	84,7	69,2	100,0	81,8
Vererbung	66	89,7	78,8	83,9	89,1	86,4	87,7
Abhängigkeit	17	71,4	29,4	41,7	69,2	52,9	60,0
Realisierung	32	67,9	59,4	63,3	75,9	68,8	72,1
Ger. Assoziation	29	48,6	58,6	53,1	78,9	51,7	62,5
Unger. Assoziation	28	87,5	50,0	63,6	79,2	67,9	73,1
Kanten (gesamt)	236	77,0	65,0	70,5	80,6	73,5	76,9

Tabelle 6.3.: Häufigkeit N, Genauigkeit P (in %), Trefferquote R (in %) und F_1-Maß (in %) aller Elemente des Testsatzes von Version 4

Das zugehörige F_1-Maß ist folglich nahezu maximal mit 99,8% auf dem Validierungssatz beziehungsweise 99,1% auf dem Testsatz.

Für die Evaluation der Erkennung von Textblöcken, wurde in dieser Arbeit lediglich die Bounding-Box dieser berücksichtigt. Eine Erkennung von Inhalt oder das Klassifizieren einer Textkategorie ist mit dem derzeitigen DiagramNet-Verfahren nicht darstellbar. Wie die Ergebnisse zeigen, können die Bounding-Boxen der Textinhalte ebenfalls relativ akkurat erkannt werden. Diese sind jedoch mit einem F_1-Maß von 92,9% beim Validierungssatz, beziehungsweise 87,8% beim Testsatz, nicht auf dem Level der Klassenknoten. Begründbar ist dies durch für das Verfahren unvermeidbare Fehler. Diese können beispielsweise beim Erkennen von Multiplizitäten auftreten, deren Bounding-Box meist nur wenige Pixel groß ist. Eine minimale Abweichung der erkannten Bounding-Box im Vergleich zur Annotation führt hier aufgrund einer zu geringen IoU bereits zu einer falsch positiven Erkennung. Des Weiteren ist es für das Verfahren nicht immer möglich, Textblöcke innerhalb eines Klassenknotens korrekt zu separieren. Bei Attributen oder Methoden können diese sowohl ausschließlich eine als auch mehrere Zeilen umfassen. Eine korrekte Trennung ist für das System ohne Interpretation des Textinhaltes nicht immer darstellbar. Mit einer expliziten Erkennung von handgeschriebenem Text könnten sich die genannten Fehler vermeiden lassen.

Bei der Erkennung von Kanten ist die Genauigkeit auf beiden Sätzen signifikant höher als die Trefferquote. Die daraus resultierenden F_1-Ergebnisse sind dabei niedriger als die der Knoten und Textblöcke. Dies ist unter anderem durch die niedrigere Häufigkeit einzelner Kantenkategorien begründbar. Fehler in der Klassifikation dieser sind somit wahrscheinlicher, als noch bei Klassenknoten oder Textblöcken. Der vergleichbare Offline-Datensatz hdBPMN beinhaltet deutlich mehr Kanten der selben Kategorie. Die Präzision und Genauigkeit der Erkennung dieser ist folglich weitaus höher. Die erzielten Werte zeigen dennoch, dass eine Erkennung und Separation von UML-Kanten nach Training dieser darstellbar ist. Speziell die F_1-Ergebnisse der Vererbungs- und Kompositionskanten mit über 80% deuten auf eine präzise Erkennung dieser hin. Durch das Erweitern des Datensatzes könnte man

in Zukunft bei weiteren Kantenkategorien verbesserte Erkennungsergebnisse erwarten. Diese Vermutung wird durch eine positive Pearson-Korrelation von 0,73 zwischen der Häufigkeit und dem zugehörigen F_1-Maß aller Kanten des Testsatzes bestärkt.

Die Ergebnisse zeigen insgesamt, dass trotz der im Vergleich zum hdBPMN-Datensatz niedrigeren Elementhäufigkeiten, eine Erkennung von UML-Klassendiagrammskizzen möglich ist. Speziell die Genauigkeit und Trefferquote der Erkennung von Knoten und Textpositionen sind auf Basis der Ergebnisse verwandter Arbeiten [14, 15, 16] bereits sehr hoch. Die Erkennungsergebnisse von Kanten werden durch das Transferlernen der BPMN-Domäne verbessert und sind mit einem F_1-Maß von 71,2% beziehungsweise 76,9% ebenfalls auf einem guten Niveau. Ein Einsatz des UML-Datensatzes dieser Arbeit für das maschinelle Lernen ist somit möglich. Die Art und Weise der Annotation und das für den Klassendiagrammkontext angepasste COCO-Format eignen sich folglich für die Eingabe maschineller Lernverfahren. Über eine Interpretation von Textinhalt sind zudem noch weitere Untersuchungen des Datensatzes möglich, die im Rahmen dieser Bachelorarbeit aufgrund der Nichtexistenz eines Texterkennungssystems für handgeschriebene Schrift noch nicht evaluiert werden konnten.

7. Fazit und zukünftige Arbeiten

Die Bachelorarbeit hat einen Datensatz von UML-Klassendiagrammskizzen für maschinelle Lernverfahren vorgestellt. Die darin enthaltenen Diagramme besitzen dabei für Skizzen typische Eigenschaften wie Ungenauigkeiten in der Modellierung. Die Daten variieren außerdem bezüglich dem zugrundeliegenden Diagrammhintergrund, dem Zeichengerät, der Aufnahmequalität und der Diagrammstruktur. Der im Rahmen dieser Arbeit entstandene UML-Image-Annotator ermöglicht darüber hinaus das präzise Annotieren von Klassendiagrammskizzen. Die Diagrammskizzen können zudem mit ihren zugehörigen Annotationen in einen COCO-Datensatz transformiert werden. Die Evaluation hat bestätigt, dass sich der Datensatz für den Einsatz maschineller Lernverfahren eignet. Ein maschinelles Lernverfahren war dabei nach Training einer Teilmenge der Daten in der Lage, Knoten mit einem F_1-Maß von über 99%, Textpositionen mit einem F_1-Maß von über 87% und Kanten mit einem F_1-Maß von über 71% zu erkennen. Durch das Transferlernen der BPMN-Domäne auf die Domäne der UML-Klassendiagramme konnten insbesondere die Erkennungsergebnisse von Kanten verbessert werden.

Zukünftige Arbeiten können an den Ergebnissen dieser Arbeit anknüpfen und den UML-Datensatz durch ergänzende Methoden der Datengewinnung erweitern. Um die Vielfältigkeit des Datensatzes weiter zu erhöhen, könnten zusätzlich handgezeichnete UML-Klassendiagramme aus der Industrie erhoben werden. Eine weitere Untersuchung des Lindholmen-Datensatzes (siehe Abschnitt 4.3) ist zu diesem Zweck ebenfalls denkbar. Darüber hinaus ist ein Einsatz des Datensatzes als Eingabe weiterer maschineller Lernverfahren möglich. Speziell Verfahren mit integrierter Texterkennung könnten über eine Interpretation von Textinhalten weitere Untersuchungen auf dem Datensatz durchführen. Über das Verfügen von Textinhalt könnte beispielsweise eine Klassifikation von Text durchgeführt werden oder Domänenwissen genutzt werden, um die Präzision bei der Erkennung von UML-Elementen weiter zu erhöhen. Des Weiteren könnten sich zukünftige Arbeiten mit der Erkennung und folglich auch mit der Erstellung von Datensätzen handgezeichneter Diagramme weiterer Domänen beschäftigen. Hierbei ist ebenfalls eine Untersuchung des in dieser Arbeit nachgewiesenen Transferlernens sinnvoll, um zu überprüfen, inwiefern sich die Beobachtungen auf andere Domänen übertragen lassen.

Literatur

[1] Michel R. V. Chaudron, Werner Heijstek und Ariadi Nugroho. „How effective is UML modeling ?“ In: *Software & Systems Modeling* 11.4 (Okt. 2012), S. 571–580. ISSN: 1619-1374. DOI: 10.1007/s10270-012-0278-4.

[2] C.F.J. Lange, M.R.V. Chaudron und J. Muskens. „In practice: UML software architecture and design description“. en. In: *IEEE Software* 23.2 (März 2006), S. 40–46. ISSN: 0740-7459. DOI: 10.1109/MS.2006.50.

[3] E. Arisholm u. a. „The impact of UML documentation on software maintenance: an experimental evaluation“. In: *IEEE Transactions on Software Engineering* 32.6 (2006), S. 365–381. DOI: 10.1109/TSE.2006.59.

[4] Steve Cook u. a. *Unified Modeling Language (UML) Version 2.5.1*. Dez. 2017. URL: https://www.omg.org/spec/UML/2.5.1 (besucht am 18. 07. 2021).

[5] D. Budgen u. a. „Empirical evidence about the UML: a systematic literature review“. en. In: *Software: Practice and Experience* 41.4 (Apr. 2011), S. 363–392. ISSN: 00380644. DOI: 10.1002/spe.1009.

[6] Marian Petre. „UML in practice“. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, S. 722–731. DOI: 10.1109/ICSE.2013.6606618.

[7] Martin Grossman, Jay E. Aronson und Richard V. McCarthy. „Does UML make the grade? Insights from the software development community“. In: *Information and Software Technology* 47.6 (2005), S. 383–397. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2004.09.005.

[8] Sparx Systems. *Enterprise Architect*. URL: https://www.sparxsystems.eu/ (besucht am 05. 08. 2021).

[9] MKLab. *StarUML*. URL: http://staruml.io/ (besucht am 03. 08. 2021).

[10] Microsoft. *Microsoft Visio*. URL: https://products.office.com/en/visio/flowchart-software (besucht am 03. 09. 2021).

[11] Arnaud Rosques. *PlantUML*. URL: https://plantuml.com/ (besucht am 03. 09. 2021).

[12] Mauro Cherubini u. a. „Let's Go to the Whiteboard: How and Why Software Developers Use Drawings“. In: New York, NY, USA: Association for Computing Machinery, 2007, S. 557–566. ISBN: 9781595935939.

[13] Frank D. Julca-Aguilar und Nina S. T. Hirata. „Symbol Detection in Online Handwritten Graphics Using Faster R-CNN“. en. In: *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*. Vienna: IEEE, Apr. 2018, S. 151–156. ISBN: 978-1-5386-3346-5. DOI: 10.1109/DAS.2018.79.

[14] Bernhard Schäfer, Margret Keuper und Heiner Stuckenschmidt. „Arrow R-CNN for handwritten diagram recognition“. en. In: *International Journal on Document Analysis and Recognition (IJDAR)* 24.1-2 (Juni 2021), S. 3–17. ISSN: 1433-2833, 1433-2825. DOI: 10.1007/s10032-020-00361-1.

[15] Bernhard Schäfer u. a. „Sketch2BPMN: Automatic Recognition of Hand-Drawn BPMN Models“. en. In: *Advanced Information Systems Engineering*. Hrsg. von Marcello La Rosa, Shazia Sadiq und Ernest Teniente. Bd. 12751. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, S. 344–360. DOI: 10.1007/978-3-030-79382-1_21.

[16] Bernhard Schäfer und Heiner Stuckenschmidt. „DiagramNet: Hand-Drawn Diagram Recognition Using Visual Arrow-Relation Detection“. en. In: *Document Analysis and Recognition – ICDAR 2021*. Cham: Springer International Publishing, Sep. 2021, S. 614–630. ISBN: 978-3-030-86549-8.

[17] Tsung-Yi Lin u. a. „Microsoft COCO: Common Objects in Context“. en. In: *arXiv:1405.0312 [cs]* (Feb. 2015). URL: http://arxiv.org/abs/1405.0312 (besucht am 19. 07. 2021).

[18] Tsung-Yi Lin u. a. *COCO Data format*. URL: https://cocodataset.org/#format-data (besucht am 19. 07. 2021).

[19] Martin Bresler, Daniel Průša und Václav Hlaváč. „Recognizing Off-Line Flowcharts by Reconstructing Strokes and Using On-Line Recognition Techniques“. In: *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 2016, S. 48–53. DOI: 10.1109/ICFHR.2016.0022.

[20] Florian Huber und Georg Hagel. „Work-in-Progress: Towards detection and syntactical analysis in UML class diagrams for software engineering education“. In: *2020 IEEE Global Engineering Education Conference (EDUCON)*. 2020, S. 3–7. DOI: 10.1109/EDUCON45650.2020.9125244.

[21] E. Lank u. a. „On-line recognition of UML diagrams“. In: *Proceedings of Sixth International Conference on Document Analysis and Recognition*. 2001, S. 356–360. DOI: 10.1109/ICDAR.2001.953813.

[22] Tracy Hammond und Randall Davis. „Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams“. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH '06. Boston, Massachusetts: Association for Computing Machinery, 2006, 25–es. ISBN: 1595933646. DOI: 10.1145/1185657.1185786.

[23] Vincenzo Deufemia und Michele Risi. „Multi-Domain Recognition of Hand-Drawn Diagrams Using Hierarchical Parsing“. In: *Multimodal Technologies and Interaction* 4.3 (2020). ISSN: 2414-4088. DOI: 10.3390/mti4030052.

[24] Martin Bresler, Daniel Průša und Václav Hlaváč. „Online recognition of sketched arrow-connected diagrams“. In: *International Journal on Document Analysis and Recognition (IJDAR)* 19.3 (Sep. 2016), S. 253–267. ISSN: 1433-2825. DOI: 10.1007/s10032-016-0269-z.

[25] Shaoqing Ren u. a. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), S. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[26] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection“. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, S. 779–788. DOI: 10.1109/CVPR.2016.91.

[27] *Data and Web Sience Group* (dwslab) der Universität Mannheim. *hdBPMN*. 2020. URL: https://github.com/dwslab/hdBPMN (besucht am 23. 11. 2021).

[28] *Business Process Model and Notation (BPMN) Version 2.0*. Jan. 2011. URL: https://www.omg.org/spec/BPMN/2.0/ (besucht am 20. 12. 2021).

[29] Bilal Karasneh und Michel RV Chaudron. „Online Img2UML Repository: An Online Repository for UML Models.“ In: *EESSMOD@ MoDELS*. Citeseer. 2013, S. 61–66.

[30] Bilal Karasneh und Michel R.V. Chaudron. „Img2UML: A System for Extracting UML Models from Images“. In: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. 2013, S. 134–137. DOI: 10.1109/SEAA.2013.45.

[31] Gregorio Robles u. a. „An Extensive Dataset of UML Models in GitHub“. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017, S. 519–522. DOI: 10.1109/MSR.2017.48.

[32] Ahmad-Montaser Awal u. a. „First experiments on a new online handwritten flowchart database“. In: *Document Recognition and Retrieval XVIII*. Hrsg. von Gady Agam und Christian Viard-Gaudin. Bd. 7874. International Society for Optics und Photonics. SPIE, 2011, S. 81–90. DOI: 10.1117/12.876624.

[33] *Data and Web Sience Group* (dwslab) der Universität Mannheim. *BPMN Image Annotator*. 2020. URL: https://github.com/dwslab/bpmn-image-annotator (besucht am 29. 11. 2021).

[34] *Data and Web Sience Group* (dwslab) der Universität Mannheim. *pybpmn*. 2021. URL: https://github.com/dwslab/pybpmn (besucht am 03. 08. 2021).

[35] *UML-Image-Annotator*. 2021. URL: https://github.com/LiSSA-Approach/uml-image-annotator (besucht am 21. 12. 2021).

[36] Camunda. *bpmn-js*. URL: https://github.com/bpmn-io/bpmn-js (besucht am 07. 08. 2021).

[37] *pybpmn-uml*. 2021. URL: https://github.com/LiSSA-Approach/pybpmn-uml (besucht am 21. 12. 2021).

[38] Walter F. Tichy, Thomas Karcher und David Meder. *Klausur Softwaretechnik 1, Karlsruher Institut für Technologie*. Okt. 2011.

[39] Walter F. Tichy, Thomas Karcher und David Meder. *Klausur Softwaretechnik 1, Karlsruher Institut für Technologie*. Aug. 2011.

[40] Walter F. Tichy, A. Höfer und David Meder. *Klausur Softwaretechnik 1, Karlsruher Institut für Technologie*. Aug. 2009.

[41] Walter F. Tichy, Christopher Gerking und Tobias Hey. *Übung Softwaretechnik 1, Karlsruher Institut für Technologie*. 2021.

A. Anhang

Schiffe sind entweder Passagier- oder Containerschiffe. Ein Passagierschiff kann ein oder mehrere Restaurants haben, welche jeweils eine bestimmte Anzahl an Sitzplätzen zur Verfügung stellen. Jedes Schiff hat mindestens einen Dieselmotor. Ein Dieselmotor besteht aus mehreren Zylindern. Jeder Zylinder hat Ventile und einen Brennraum.

Modellieren Sie das Szenario möglichst vollständig als UML-Klassendiagramm.

Abbildung A.1.: Erstes Ursprungsszenario der Nutzerstudie, stammend aus [38]

Ein Güterzug ist ein Zug, dessen Waggons ausschließlich Güterwagen sind. Die Waggons eines Personenzugs sind mindestens ein Reisezugwagen und höchstens ein Speisewagen. Jeder Zug hat eine oder zwei Lokomotiven. Reisezugwagen setzen sich aus bis zu einem Großraum sowie einem oder mehreren Abteilen zusammen. Jeder Reisezugwagen hat eine Klimaanlage, die ein- und ausgeschaltet werden kann. Jede Klimaanlage darf nur bis zu einer bestimmten maximalen Außentemperatur betrieben werden.

Modellieren Sie das Szenario möglichst vollständig als UML-Klassendiagramm. Geben Sie Methoden, Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen, Aggregationen und Kompositionen sowie Rollen an.

Abbildung A.2.: Zweites Ursprungsszenario der Nutzerstudie, stammend aus [39]

Ein Fachwerkhaus besteht aus 5 bis 10 Holzstämmen, 200 bis 400 Lehmziegeln sowie 1.000 bis 2.000 Nägeln. Jedes Baumaterial, egal ob Holzstamm, Lehmziegel oder Nagel, ist Bestandteil in genau einem Fachwerkhaus. Jedes Fachwerkhaus hat eine bestimmte Anzahl an Zimmern und Stockwerken. Für den Bau eines Fachwerkhauses ist mindestens ein Zimmermann zuständig, welcher einen Namen sowie einen individuellen Stundenlohn besitzt. Zum Bau des Fachwerkhauses verwendet jeder Zimmermann sein eigenes Werkzeug, bestehend aus genau einem Hammer sowie genau einer Säge. Jeder Zimmermann kann an maximal einem Fachwerkhaus gleichzeitig bauen.

Modellieren Sie das Szenario möglichst vollständig als UML-Klassendiagramm. Modellieren Sie keine Methoden. Geben Sie Attribute, Multiplizitäten, Restriktionen, Assoziationsnamen sowie Rollen an. (9 P)

Abbildung A.3.: Drittes Ursprungsszenario der Nutzerstudie, stammend aus [40]

Aufgabe 3: Klassendiagramm (4 + 3 Punkte)

Teilnahme an Datensammlung für Forschungsprojekt
Skizzen sind das Mittel der Wahl für eine einfache Kommunikation zwischen Softwareentwicklern oder Softwarearchitekten. Maschinelle Lernansätze benötigen für ihre Funktionalität jedoch viele Trainingsdaten. Daher nun die Bitte, um Ihre Mithilfe an einem aktuellen Forschungsthema: Sofern Sie damit einverstanden sind, würde Herr Fuchß gerne Ihre Skizzen zu dieser Aufgabe für seine aktuelle Forschung verwenden.
Wofür und wie werden Ihre Skizzen verwendet:

- Ihre Skizzen sollen im Rahmen einer Abschlussarbeit analysiert und dazu verwendet werden, ein Modell zu trainieren, das Skizzen von UML-Diagrammen automatisch erkennen kann.
- Ihre Skizzen sollen Teil eines *anonymisierten* Datensatzes werden, welcher in einer Publikation veröffentlicht werden soll, damit Forscher weltweit einen gemeinsamen Trainingsdatensatz verwenden können.

Sofern Sie mit dem Verwenden auf diese Art und Weise einverstanden sind, schreiben Sie dies bitte einfach als formlosen Kommentar auf das Blatt Ihrer Lösung:
z.B. "Ich bin mit der anonymisierten Verwendung meiner Diagramme, die ich als Übungsaufgabe zur Vorlesung SWT1 erstellt habe, zu Forschungszwecken einverstanden."
Sollten Sie nicht einverstanden sein müssen Sie nichts weiter tun und Ihre Abgabe wird nicht verwendet.
Für Nachfragen steht Herr Fuchß Ihnen jederzeit per Mail zur Verfügung [dominik.fuchss@kit.edu]

Ihre Projektleiterin wurde vom Vorstand der Pear Corp. mit der Umsetzung des Projektes iMage beauftragt. Da ihre anstehende Beförderung vom Erfolg des Projektes abhängt, möchte sie, dass zunächst der aktuelle Zustand des Systems als UML-Klassendiagramm festgehalten wird. Zur Erinnerung: iMage basiert auf JMJRST (siehe Übungsblatt 1).
Beachten Sie: Bitte verwenden Sie zur Bearbeitung der Aufgabenteile a) und b) unterschiedlich-farbige Stifte (z.B. blau für Teil a) und schwarz für Teil b)). Verwenden Sie keine roten Stifte.

a) Erstellen Sie basierend auf der folgenden Beschreibung ein UML-Klassendiagramm:

> *iMage enthält verschiedene Verfahren. iMage unterscheidet bei den Verfahren zwischen Verfahren die auf ein einzelnes Bild angewendet werden und Verfahren für mehrere Bilder. Bei Einzelbildverfahren wird eine Verarbeitung auf einem Bild durchgeführt und das resultierende Bild zurückgegeben. Gegebene Einzelbildverfahren sind einerseits die Qualitätsreduktion, welche das Bild auf eine festgelegte Farbtiefe der originalen Qualität reduziert, sowie andererseits die Skalierung, bei der das Bild auf eine festgelegte Höhe und Breite skaliert wird. In vorangehenden Arbeiten der Pear Corp. wurden bereits Mehrbildverfahren zur Collagenerstellung und zur Mosaikerstellung umgesetzt. Mehrbildverfahren erhalten bei jeder Verarbeitung eine Menge von Bildern. Auf diesen führen sie ihr Verfahren aus und geben das Ergebnis als neues Bild zurück. iMage verwendet RGBA-Bilder, welche aus einer beliebigen Anzahl an Pixeln bestehen. Jeder Pixel hat eine Farbe, bestehend aus vier ganzzahligen Farbkanälen (rot (R), grün (G), blau (B) und alpha (A)). Der alpha-Kanal gibt hierbei die Transparenz des Pixels an. Außerdem können Bilder in unterschiedlichen Farbtiefen vorliegen (8-Bit, 24-Bit, 32-Bit).*

Abbildung A.4.: Aufgabenstellung der Studierendenabgabe (1/2), stammend aus [41]

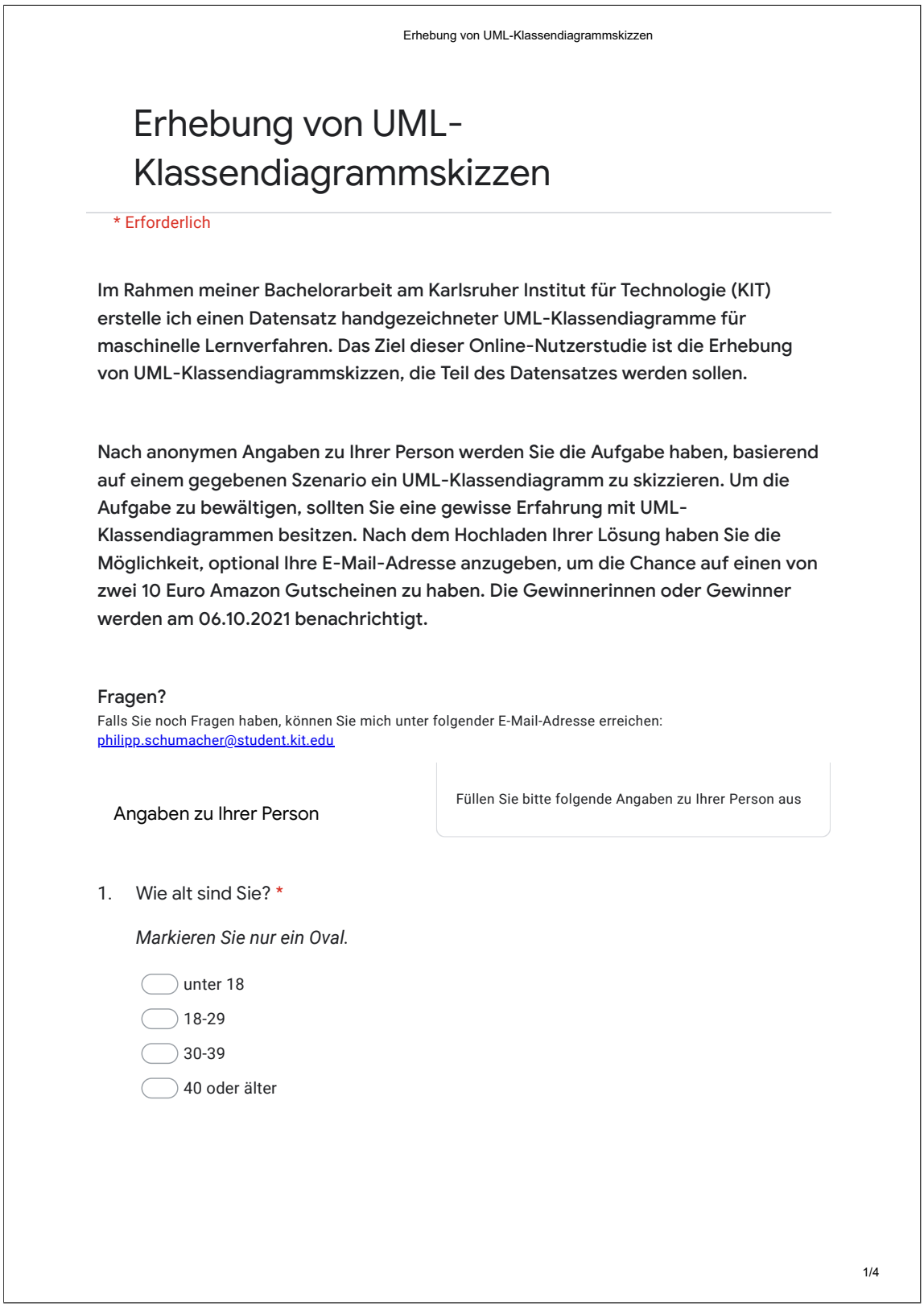
Nun soll iMage so erweitert werden, dass ein neues Mehrbildverfahren *ScreenGen* hinzugefügt wird, welches eine bestimmte Farbe aus einem Bild entfernt und ein anderes Bild in dessen Hintergrund einfügt.

b) Erweitern Sie Ihr UML-Klassendiagramm gemäß der nachfolgenden Beschreibung:

> *ScreenGen ist ein Mehrbildverfahren, welches aus zwei Einzelbildverfahren besteht und mithilfe dieser zwei Verfahren ein Ergebnisbild erstellt. Prinzipiell soll ScreenGen mit beliebigen Einzelbildverfahren arbeiten können. Zunächst werden die folgenden zwei Einzelbildverfahren nacheinander verwendet: Das erste Einzelbildverfahren bezieht sich auf eine Farbe. Das Verfahren bestimmt diejenigen Pixel des ersten Eingabebildes, die dieser Farbe entsprechen, und setzt den Farbwert dieser Pixel auf transparent. Das zweite Einzelbildverfahren nutzt die Bibliothek ImageUtils um ein zweites Eingabebild in den Hintergrund des freigestellten Bildes zu setzen (es existieren jedoch keine Instanzen von ImageUtils). Dazu bezieht sich das Verfahren auf eine Position. Diese Position bestimmt, ob das freigestellte Bild auf dem Hintergrund entweder oben links, oben rechts, mittig oder unten mittig positioniert werden soll.*

Verwenden Sie in Ihrer Modellierung UML-Elemente des Klassendiagramms; z.B. Klassen, Schnittstellen, Vererbungs- und Implementierungsbeziehungen, Assoziationen samt Multiplizitäten, Aggregationen und Kompositionen. Verzichten Sie auf Nutzung der Object Constraint Language (OCL). Reichern Sie die Klassen um Attribute und Methodennamen an, soweit sie für die Modellierung der Beschreibung gebraucht werden.
Zeichnen Sie das Diagramm als Übung für die Klausur von Hand.

Abbildung A.5.: Aufgabenstellung der Studierendenabgabe (2/2), stammend aus [41]



Erhebung von UML-Klassendiagrammskizzen

Erhebung von UML-Klassendiagrammskizzen

* Erforderlich

Im Rahmen meiner Bachelorarbeit am Karlsruher Institut für Technologie (KIT) erstelle ich einen Datensatz handgezeichneter UML-Klassendiagramme für maschinelle Lernverfahren. Das Ziel dieser Online-Nutzerstudie ist die Erhebung von UML-Klassendiagrammskizzen, die Teil des Datensatzes werden sollen.

Nach anonymen Angaben zu Ihrer Person werden Sie die Aufgabe haben, basierend auf einem gegebenen Szenario ein UML-Klassendiagramm zu skizzieren. Um die Aufgabe zu bewältigen, sollten Sie eine gewisse Erfahrung mit UML-Klassendiagrammen besitzen. Nach dem Hochladen Ihrer Lösung haben Sie die Möglichkeit, optional Ihre E-Mail-Adresse anzugeben, um die Chance auf einen von zwei 10 Euro Amazon Gutscheinen zu haben. Die Gewinnerinnen oder Gewinner werden am 06.10.2021 benachrichtigt.

Fragen?
Falls Sie noch Fragen haben, können Sie mich unter folgender E-Mail-Adresse erreichen:
philipp.schumacher@student.kit.edu

Angaben zu Ihrer Person

Füllen Sie bitte folgende Angaben zu Ihrer Person aus

1. Wie alt sind Sie? *

Markieren Sie nur ein Oval.

- unter 18
- 18-29
- 30-39
- 40 oder älter

1/4

Abbildung A.6.: Google Formular 1 (1/4)

2. Sie sind... *

Markieren Sie nur ein Oval.

- Männlich
- Weiblich
- Divers
- keine Angabe

3. Was ist Ihr höchster Bildungsabschluss? *

Markieren Sie nur ein Oval.

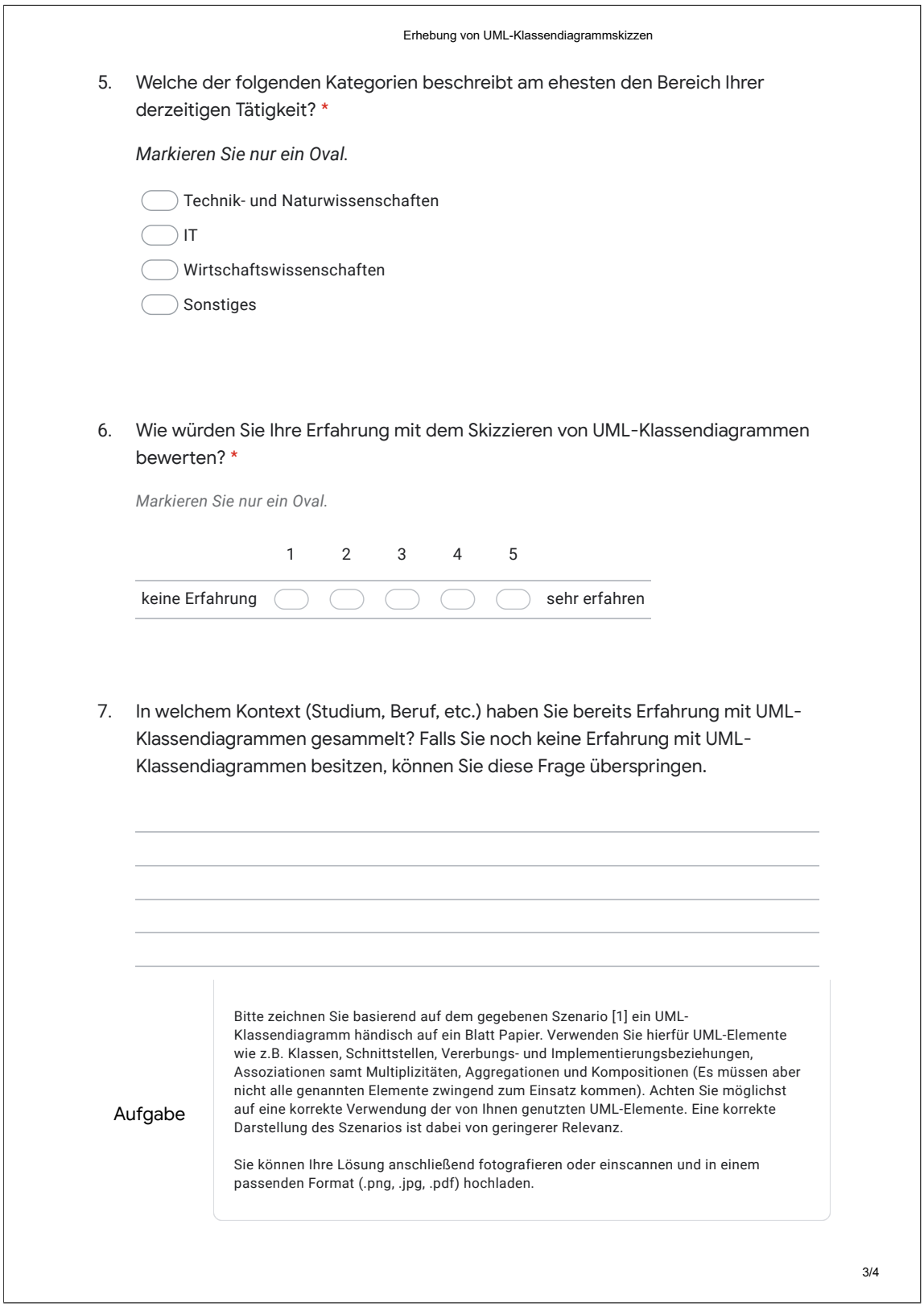
- kein Abschluss
- Grund-/Hauptschulabschluss
- Mittlere Reife
- Fachhochschulreife/Abitur
- Abgeschlossene Ausbildung
- Abgeschlossenes Studium (Bachelor)
- Abgeschlossenes Studium (Master)
- keine Angabe

4. Welche der folgenden Kategorien beschreibt am ehesten Ihre derzeitige Tätigkeit? *

Markieren Sie nur ein Oval.

- Ausbildung
- Studium
- Berufstätig
- Sonstiges

Abbildung A.7.: Google Formular 1 (2/4)



Erhebung von UML-Klassendiagrammskizzen

5. Welche der folgenden Kategorien beschreibt am ehesten den Bereich Ihrer derzeitigen Tätigkeit? *

Markieren Sie nur ein Oval.

- Technik- und Naturwissenschaften
- IT
- Wirtschaftswissenschaften
- Sonstiges

6. Wie würden Sie Ihre Erfahrung mit dem Skizzieren von UML-Klassendiagrammen bewerten? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
keine Erfahrung						sehr erfahren

7. In welchem Kontext (Studium, Beruf, etc.) haben Sie bereits Erfahrung mit UML-Klassendiagrammen gesammelt? Falls Sie noch keine Erfahrung mit UML-Klassendiagrammen besitzen, können Sie diese Frage überspringen.

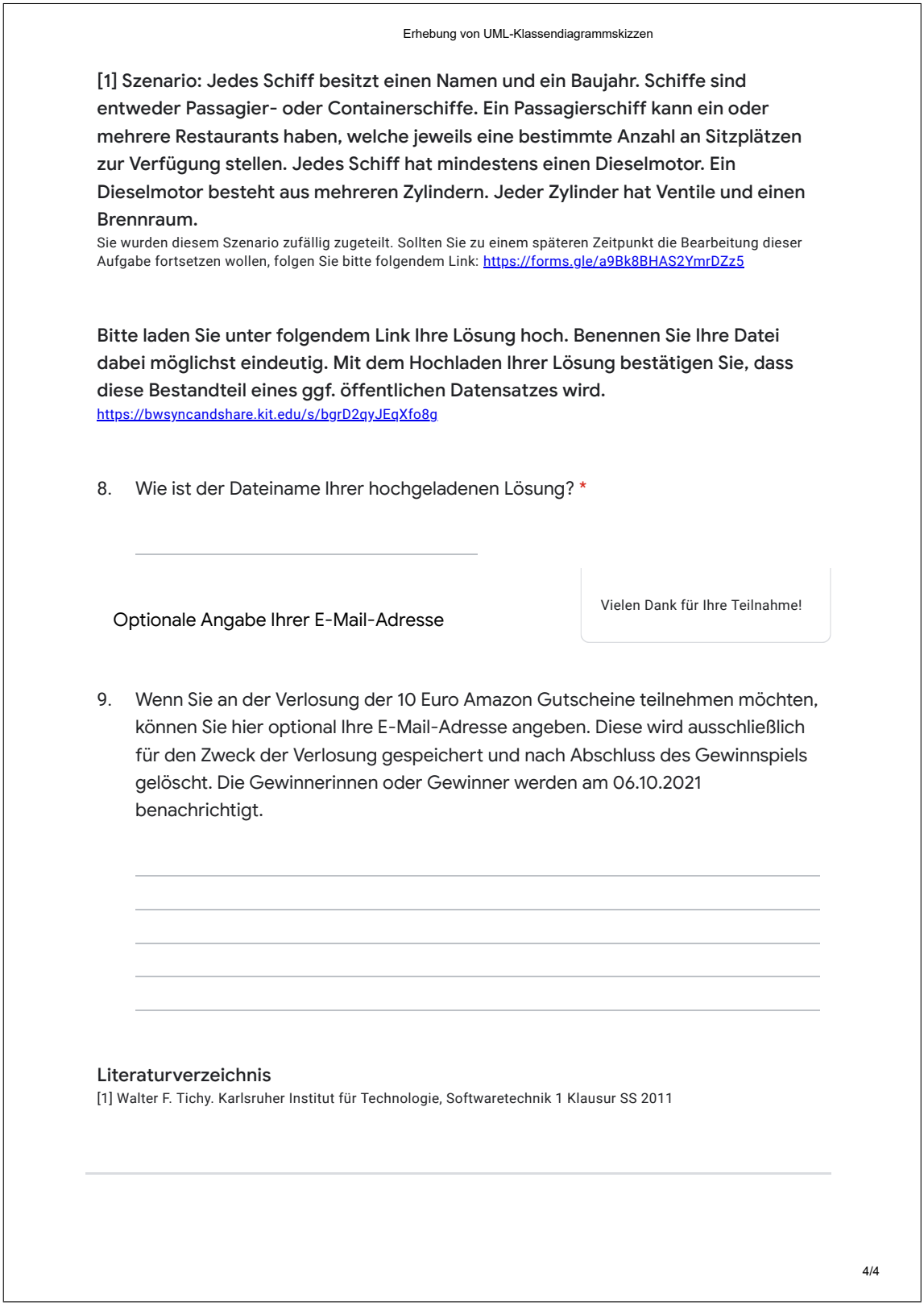
Aufgabe

Bitte zeichnen Sie basierend auf dem gegebenen Szenario [1] ein UML-Klassendiagramm händisch auf ein Blatt Papier. Verwenden Sie hierfür UML-Elemente wie z.B. Klassen, Schnittstellen, Vererbungs- und Implementierungsbeziehungen, Assoziationen samt Multiplizitäten, Aggregationen und Kompositionen (Es müssen aber nicht alle genannten Elemente zwingend zum Einsatz kommen). Achten Sie möglichst auf eine korrekte Verwendung der von Ihnen genutzten UML-Elemente. Eine korrekte Darstellung des Szenarios ist dabei von geringerer Relevanz.

Sie können Ihre Lösung anschließend fotografieren oder einscannen und in einem passenden Format (.png, .jpg, .pdf) hochladen.

3/4

Abbildung A.8.: Google Formular 1 (3/4)



Erhebung von UML-Klassendiagrammskizzen

[1] Szenario: Jedes Schiff besitzt einen Namen und ein Baujahr. Schiffe sind entweder Passagier- oder Containerschiffe. Ein Passagierschiff kann ein oder mehrere Restaurants haben, welche jeweils eine bestimmte Anzahl an Sitzplätzen zur Verfügung stellen. Jedes Schiff hat mindestens einen Dieselmotor. Ein Dieselmotor besteht aus mehreren Zylindern. Jeder Zylinder hat Ventile und einen Brennraum.

Sie wurden diesem Szenario zufällig zugeteilt. Sollten Sie zu einem späteren Zeitpunkt die Bearbeitung dieser Aufgabe fortsetzen wollen, folgen Sie bitte folgendem Link: https://forms.gle/a9Bk8BHAS2YmrDZz5

Bitte laden Sie unter folgendem Link Ihre Lösung hoch. Benennen Sie Ihre Datei dabei möglichst eindeutig. Mit dem Hochladen Ihrer Lösung bestätigen Sie, dass diese Bestandteil eines ggf. öffentlichen Datensatzes wird.
https://bwsyncandshare.kit.edu/s/bgrD2qyJEqXfo8g

8. Wie ist der Dateiname Ihrer hochgeladenen Lösung? *

Optionale Angabe Ihrer E-Mail-Adresse

Vielen Dank für Ihre Teilnahme!

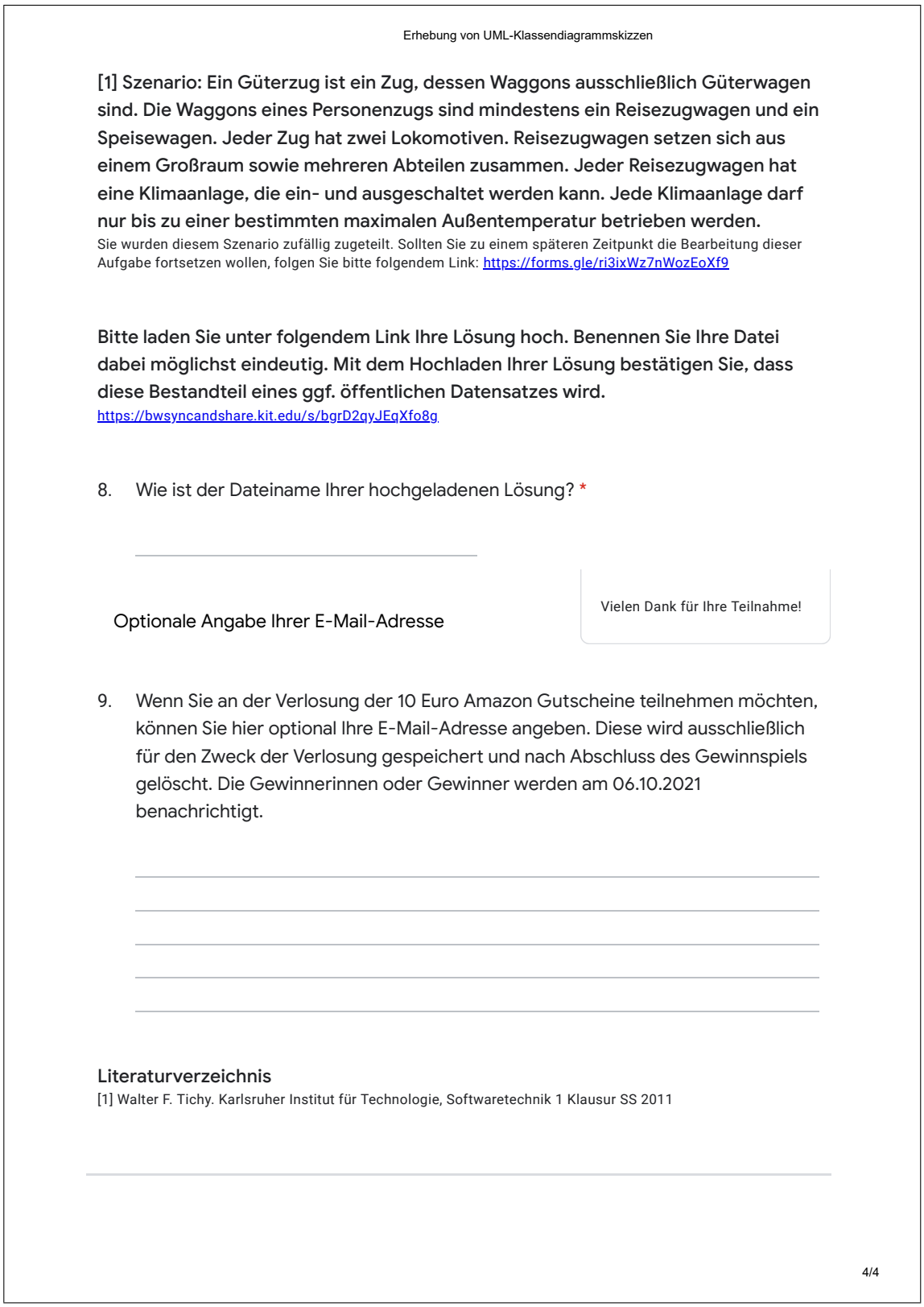
9. Wenn Sie an der Verlosung der 10 Euro Amazon Gutscheine teilnehmen möchten, können Sie hier optional Ihre E-Mail-Adresse angeben. Diese wird ausschließlich für den Zweck der Verlosung gespeichert und nach Abschluss des Gewinnspiels gelöscht. Die Gewinnerinnen oder Gewinner werden am 06.10.2021 benachrichtigt.

Literaturverzeichnis

[1] Walter F. Tichy. Karlsruher Institut für Technologie, Softwaretechnik 1 Klausur SS 2011

4/4

Abbildung A.9.: Google Formular 1 (4/4): Szenario



Erhebung von UML-Klassendiagrammskizzen

[1] Szenario: Ein Güterzug ist ein Zug, dessen Waggons ausschließlich Güterwagen sind. Die Waggons eines Personenzugs sind mindestens ein Reisezugwagen und ein Speisewagen. Jeder Zug hat zwei Lokomotiven. Reisezugwagen setzen sich aus einem Großraum sowie mehreren Abteilen zusammen. Jeder Reisezugwagen hat eine Klimaanlage, die ein- und ausgeschaltet werden kann. Jede Klimaanlage darf nur bis zu einer bestimmten maximalen Außentemperatur betrieben werden.

Sie wurden diesem Szenario zufällig zugeteilt. Sollten Sie zu einem späteren Zeitpunkt die Bearbeitung dieser Aufgabe fortsetzen wollen, folgen Sie bitte folgendem Link: https://forms.gle/ri3ixWz7nWozEoXf9

Bitte laden Sie unter folgendem Link Ihre Lösung hoch. Benennen Sie Ihre Datei dabei möglichst eindeutig. Mit dem Hochladen Ihrer Lösung bestätigen Sie, dass diese Bestandteil eines ggf. öffentlichen Datensatzes wird.

https://bwsyncandshare.kit.edu/s/bgrD2qyJEqXfo8g

8. Wie ist der Dateiname Ihrer hochgeladenen Lösung? *

Optionale Angabe Ihrer E-Mail-Adresse

Vielen Dank für Ihre Teilnahme!

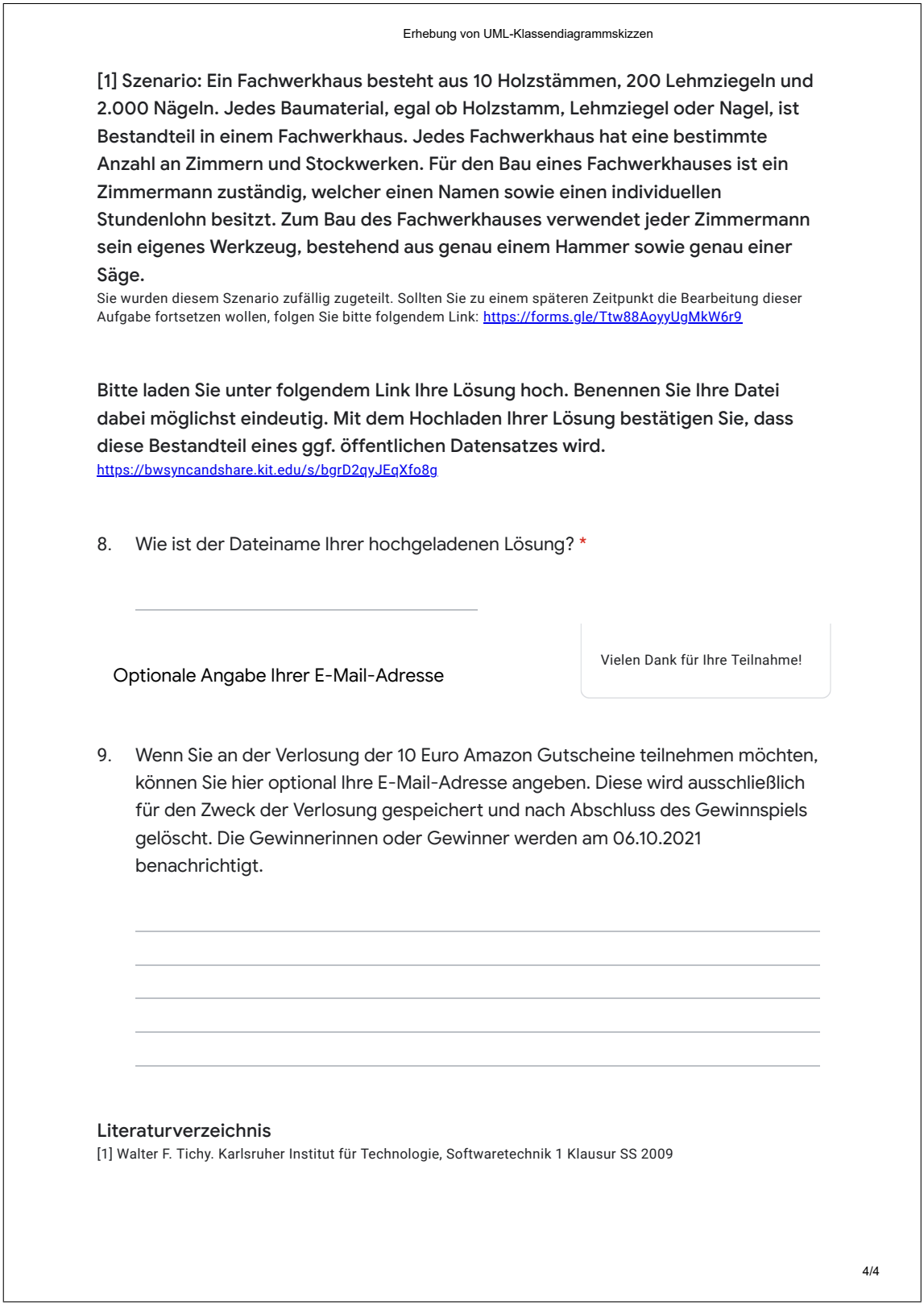
9. Wenn Sie an der Verlosung der 10 Euro Amazon Gutscheine teilnehmen möchten, können Sie hier optional Ihre E-Mail-Adresse angeben. Diese wird ausschließlich für den Zweck der Verlosung gespeichert und nach Abschluss des Gewinnspiels gelöscht. Die Gewinnerinnen oder Gewinner werden am 06.10.2021 benachrichtigt.

Literaturverzeichnis

[1] Walter F. Tichy. Karlsruher Institut für Technologie, Softwaretechnik 1 Klausur SS 2011

4/4

Abbildung A.10.: Google Formular 2: Szenario



Erhebung von UML-Klassendiagrammskizzen

[1] Szenario: Ein Fachwerkhaus besteht aus 10 Holzstämmen, 200 Lehmziegeln und 2.000 Nägeln. Jedes Baumaterial, egal ob Holzstamm, Lehmziegel oder Nagel, ist Bestandteil in einem Fachwerkhaus. Jedes Fachwerkhaus hat eine bestimmte Anzahl an Zimmern und Stockwerken. Für den Bau eines Fachwerkhauses ist ein Zimmermann zuständig, welcher einen Namen sowie einen individuellen Stundenlohn besitzt. Zum Bau des Fachwerkhauses verwendet jeder Zimmermann sein eigenes Werkzeug, bestehend aus genau einem Hammer sowie genau einer Säge.

Sie wurden diesem Szenario zufällig zugeteilt. Sollten Sie zu einem späteren Zeitpunkt die Bearbeitung dieser Aufgabe fortsetzen wollen, folgen Sie bitte folgendem Link: https://forms.gle/Ttw88AoyyUgMkW6r9

Bitte laden Sie unter folgendem Link Ihre Lösung hoch. Benennen Sie Ihre Datei dabei möglichst eindeutig. Mit dem Hochladen Ihrer Lösung bestätigen Sie, dass diese Bestandteil eines ggf. öffentlichen Datensatzes wird.
https://bwsyncandshare.kit.edu/s/bgrD2qyJEqXfo8g

8. Wie ist der Dateiname Ihrer hochgeladenen Lösung? *

Optionale Angabe Ihrer E-Mail-Adresse

Vielen Dank für Ihre Teilnahme!

9. Wenn Sie an der Verlosung der 10 Euro Amazon Gutscheine teilnehmen möchten, können Sie hier optional Ihre E-Mail-Adresse angeben. Diese wird ausschließlich für den Zweck der Verlosung gespeichert und nach Abschluss des Gewinnspiels gelöscht. Die Gewinnerinnen oder Gewinner werden am 06.10.2021 benachrichtigt.

Literaturverzeichnis

[1] Walter F. Tichy. Karlsruher Institut für Technologie, Softwaretechnik 1 Klausur SS 2009

4/4

Abbildung A.11.: Google Formular 3: Szenario

Elementart	N	Mittelwert	Min.	Max.	Standardabw.
Klasse	1050	10,8	3	22	3,3
Schnittstelle	87	0,9	0	4	1,1
abstrakte Klasse	106	1,1	0	4	1,4
Enumeration	96	1,0	0	2	0,9
Qualifizierer	9	0,1	0	2	0,3
Paket	5	0,1	0	1	0,2
Raute aus mehrgl. Assoziation	5	0,1	0	2	0,3
Notiz	13	0,1	0	5	0,6
Objekt	15	0,2	0	7	1,0
Utility	6	0,1	0	1	0,2
Bibliothek	5	0,1	0	1	0,2
Knoten (gesamt)	1397	14,4	5	26	4,1
Gerichtete Assoziation	242	2,5	0	14	3,2
Ungerichtete Assoziation	189	1,9	0	11	2,6
Aggregation	153	1,6	0	7	1,9
Komposition	147	1,5	0	7	1,3
Abhängigkeit	116	1,2	0	11	2,5
Vererbung	475	4,9	0	15	4,0
Realisierung	186	1,9	0	11	3,0
Notizverbindung	14	0,1	0	6	0,7
Kanten (gesamt)	1522	15,7	0	34	6,0
Elementname	1382	14,2	5	25	3,9
Attribut	859	8,9	0	26	5,0
Methode	825	8,5	0	37	6,7
Enumerationswert	335	3,5	0	7	3,2
Kantenbeschriftung	653	6,7	0	34	6,0
Multiplizität	872	9,0	0	33	6,5
Kommentar	15	0,2	0	5	0,7
Textblöcke (gesamt)	4941	50,9	22	101	17,1

Tabelle A.1.: Studierendenabgabe: Detaillierte Statistik der in den Skizzen auftretenden Elementarten mit Häufigkeit N, Mittelwert, Minimum, Maximum und Standardabweichung

Elementart	N	Mittelwert	Min.	Max.	Standardabw.
Klasse	154	8,6	3	11	2,0
Schnittstelle	1	0,1	0	1	0,2
abstrakte Klasse	1	0,1	0	1	0,2
Enumeration	0	0,0	0	0	0,0
Qualifizierer	0	0,0	0	0	0,0
Paket	0	0,0	0	0	0,0
Raute aus mehrgl. Assoziation	0	0,0	0	0	0,0
Notiz	0	0,0	0	0	0,0
Objekt	0	0,0	0	0	0,0
Utility	0	0,0	0	0	0,0
Bibliothek	0	0,0	0	0	0,0
Knoten (gesamt)	156	8,7	3	11	1,9
Gerichtete Assoziation	11	0,6	0	3	1,1
Ungerichtete Assoziation	34	1,9	0	7	2,3
Aggregation	25	1,4	0	6	1,8
Komposition	24	1,3	0	5	1,4
Abhängigkeit	5	0,3	0	5	1,1
Vererbung	54	3,0	0	5	1,7
Realisierung	2	0,1	0	2	0,5
Notizverbindung	0	0,0	0	0	0,0
Kanten (gesamt)	155	8,6	2	13	2,8
Elementname	156	8,7	3	11	1,9
Attribut	75	4,2	1	15	3,4
Methode	19	1,1	0	8	2,0
Enumerationswert	0	0,0	0	0	0,0
Kantenbeschriftung	22	1,2	0	8	2,2
Multiplizität	122	6,8	0	16	5,2
Kommentar	0	0,0	0	0	0,0
Textblöcke (gesamt)	394	21,9	12	37	6,6

Tabelle A.2.: Nutzerstudie: Detaillierte Statistik der in den Skizzen auftretenden Elementarten mit Häufigkeit N, Mittelwert, Minimum, Maximum und Standardabweichung

Elementart	N	Mittelwert	Min.	Max.	Standardabw.
Klasse	1204	10,5	3	22	3,2
Schnittstelle	88	0,8	0	4	1,0
abstrakte Klasse	107	0,9	0	4	1,3
Enumeration	96	0,8	0	2	0,9
Qualifizierer	9	0,1	0	2	0,3
Paket	5	0,0	0	1	0,2
Raute aus mehrgl. Assoziation	5	0,0	0	2	0,2
Notiz	13	0,1	0	5	0,6
Objekt	15	0,1	0	7	0,9
Utility	6	0,1	0	1	0,2
Bibliothek	5	0,0	0	1	0,2
Knoten (gesamt)	1553	13,5	3	26	4,4
Gerichtete Assoziation	253	2,2	0	14	3,0
Ungerichtete Assoziation	223	1,9	0	11	2,6
Aggregation	178	1,5	0	7	1,9
Komposition	171	1,5	0	7	1,3
Abhängigkeit	121	1,1	0	11	2,4
Vererbung	529	4,6	0	15	3,8
Realisierung	188	1,6	0	11	2,8
Notizverbindung	14	0,1	0	6	0,6
Kanten (gesamt)	1677	14,6	0	34	6,2
Elementname	1538	13,4	3	25	4,2
Attribut	934	8,1	0	26	5,1
Methode	844	7,3	0	37	6,7
Enumerationswert	335	2,9	0	7	3,2
Kantenbeschriftung	675	5,9	0	34	6,0
Multiplizität	994	8,6	0	33	6,3
Kommentar	15	0,1	0	5	0,6
Textblöcke (gesamt)	5335	46,4	12	101	19,0

Tabelle A.3.: Gesamtdatensatz: Detaillierte Statistik der in den Skizzen auftretenden Elementarten mit Häufigkeit N, Mittelwert, Minimum, Maximum und Standardabweichung

Trennung	Typ	Elementart (Häufigkeit)
Training	Knoten	Klasse (840), Schnittstelle (62), Enumeration (71), Abstrakte Klasse (90), Qualifizierer (9), Paket (5), Raute aus mehrgliedriger Assoziation (5), Notiz (13), Objekt (15), Utility (5), Bibliothek (4)
	Kanten	Gerichtete Assoziation (201), Aggregation (129), Ungerichtete Assoziation (150), Komposition (118), Abhängigkeit (87), Realisierung (116), Vererbung (400), Notizverbindung (14)
	Textblöcke	Elementname (1105), Attribut (715), Methode (668), Enumerationswert (247), Kantenbeschriftung (477), Multiplizität (689), Kommentar (Text in Notizknoten) (15)
Validierung	Knoten	Klasse (186), Schnittstelle (13), Enumeration (11), Abstrakte Klasse (10), Bibliothek (1)
	Kanten	Gerichtete Assoziation (23), Aggregation (22), Ungerichtete Assoziation (45), Komposition (26), Abhängigkeit (17), Realisierung (40), Vererbung (63)
	Textblöcke	Elementname (221), Attribut (113), Methode (103), Enumerationswert (39), Kantenbeschriftung (78), Multiplizität (153)
Test	Knoten	Klasse (178), Schnittstelle (13), Enumeration (14), Abstrakte Klasse (7), Utility (1)
	Kanten	Gerichtete Assoziation (29), Aggregation (27), Ungerichtete Assoziation (28), Komposition (27), Abhängigkeit (17), Realisierung (32), Vererbung (66)
	Textblöcke	Elementname (212), Attribut (106), Methode (73), Enumerationswert (49), Kantenbeschriftung (120), Multiplizität (152)
Gesamt	Knoten	Klasse (1204), Schnittstelle (88), Enumeration (96), Abstrakte Klasse (107), Qualifizierer (9), Paket (5), Raute aus mehrgliedriger Assoziation (5), Notiz (13), Objekt (15), Utility (6), Bibliothek (5)
	Kanten	Gerichtete Assoziation (253), Aggregation (178), Ungerichtete Assoziation (223), Komposition (171), Abhängigkeit (121), Realisierung (188), Vererbung (529), Notizverbindung (14)
	Textblöcke	Elementname (1538), Attribut (934), Methode (844), Enumerationswert (335), Kantenbeschriftung (675), Multiplizität (994), Kommentar (Text in Notizknoten) (15)

Tabelle A.4.: Detaillierte Elementverteilung: Training, Validierung und Test