communicated by:
Lehr- und Forschungsgebiet Informatik 9

Prof Dr. Ulrik Schroeder



**Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet Informatik 9**
**The present work was submitted to the Learning Technologies Research Group**

# Entwicklung einer Datenstruktur zur Klassifizierung unterschiedlicher Übungsaufgaben-Elemente

Development of a Data Structure for the Classification of Different Exercise Task Elements

Masterarbeit
Master-Thesis


von / presented by

Siebes, Fabian
394747


Professor Dr. Ulrik Schroeder

Professor Dr. Jürgen Giesl

Aachen, 26. April 2024

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **iv**

**Tabellenverzeichnis** **vi**

**Quellcodeverzeichnis** **vii**

**1 Einleitung** **3**
1.1 Motivation . . . 3
1.2 Zielsetzung . . . 4
1.3 Grundlagen . . . 5
1.3.1 Project Jupyter . . . 5
1.3.2 Parsons-Problem . . . 6
1.3.3 Unterschied zwischen Aufgabenteil und Teilaufgabe . . . 7
1.4 Gliederung . . . 7

**2 Verwandte Arbeiten** **9**
2.1 Was ist Feedback? . . . 9
2.1.1 Wie sieht gutes Feedback aus? . . . 9
2.1.2 Verschiedene Arten von Feedback . . . 10
2.2 Feedback-Generierung . . . 11
2.2.1 Generelle Feedback-Generierung . . . 11
2.2.2 Feedback-Generierung mit LLMs . . . 12
2.2.3 Zusammenfassung . . . 15
2.3 Klassifizierung von Aufgaben . . . 15
2.3.1 Arten der Klassifizierung . . . 15
2.3.2 Kombinationen von Aufgabentypen . . . 20
2.3.3 Zusammenfassung . . . 20
2.4 Datenstrukturen für Aufgaben . . . 21
2.4.1 Educational Modeling Language . . . 21
2.4.2 IMS Learning Design . . . 22
2.4.3 Learning Object Metadata . . . 22
2.4.4 Sharable Content Object Reference Model . . . 22
2.4.5 ProFormA . . . 22
2.5 Warum ist eine neue Datenstruktur notwendig? . . . 23

**3 Erarbeitung der Datenstruktur** **25**
3.1 Methodik . . . 25
3.1.1 Interviews mit Erstellenden . . . 25

3.1.2 Analyse der Jupyter-Notebook-Dokumente . . . 26
3.2 Ergebnisse . . . 27
3.2.1 Interviews mit den Erstellenden der Übungsblätter . . . 27
3.2.2 Jupyter-Notebook-Dokumente . . . 28
3.3 Anforderung an eine Datenstruktur . . . 32
3.4 Datenstruktur - Erster Entwurf . . . 33
3.5 Diskussion . . . 35

**4 Entwicklung des Plugins 38**
4.1 Zweck und Anwendungsbereich . . . 38
4.2 Entwurf . . . 39
4.2.1 Anforderungen . . . 40
4.2.2 Design . . . 40
4.2.3 Integration in JupyterLab . . . 42
4.3 Funktionsübersicht . . . 43
4.3.1 Initiale Erstellung der Datenstruktur . . . 43
4.3.2 Bearbeitung der Struktur . . . 43
4.3.3 Klassifizierung der Aufgabenelemente . . . 44
4.3.4 Automatische Reparatur . . . 45
4.3.5 Sichtbarkeit des Plugins . . . 46
4.4 Implementierung . . . 46

**5 Evaluation 51**
5.1 Aufbau und Zweck der Umfrage . . . 51
5.1.1 Plugin testen . . . 51
5.1.2 Fragen beantworten . . . 52
5.2 Auswertung der Ergebnisse . . . 53
5.2.1 System Usability Scale . . . 53
5.2.2 Evaluation des Plugins . . . 55
5.2.3 Evaluation der Datenstruktur . . . 61
5.2.4 Andere Anwendungsbereiche . . . 65
5.2.5 Vergleich der klassifizierten Jupyter-Notebook-Dokumente . . . 65
5.3 Datenstruktur . . . 66
5.3.1 Änderungen an der Datenstruktur . . . 67

**6 Diskussion 70**
6.1 Analyse der Notebook-Dokumente . . . 70
6.2 Datenstruktur . . . 70
6.3 Plugin . . . 71
6.4 Evaluation . . . 72

**7 Zusammenfassung 74**

**8 Ausblick 76**
8.1 Plugin . . . 76
8.2 Datenstruktur . . . 77

**Anhänge** **79**

**A Interviewfragen** **80**

**B Ausschnitt aus dem Beispiel-Dokument für die Evaluation** **81**

**C Evaluationsfragebogen (de)** **82**

**D Evaluationsfragebogen (en)** **94**

**E XML-Schema der Datenstruktur** **105**

**F Literatur** **108**

**G Digitaler Anhang** **114**

# Abbildungsverzeichnis


1.1 Screenshot von *JupyterLab* mit einem geöffnetem *Notebook*-Dokument 6

3.1 Darstellung des XML-Schemas für einen `task` im ersten Entwurf der Datenstruktur. Das `structure`-Element wird hier gekürzt dargestellt und ist in Abbildung 3.2 vollständig zu sehen. . . . . 34

3.2 Darstellung des XML-Schemas für die Struktur der Aufgabenelemente in einem `structure`-Element im ersten Entwurf der Datenstruktur. Der Typ `QStyle` für das entsprechende Attribut bei dem `question`-Element steht für die Optionen „multiple-choice“, „write-code“, „explain-code“, „draw“ und „parsons-problem“. Das `task`-Element wird hier gekürzt dargestellt und ist in Abbildung 3.1 vollständig zu sehen. . . . . 36

4.1 Ablaufdiagramm zur Erstellung einer Datenstruktur für die Feedback-Generierung . . . . . 39

4.2 Entwurf zur Darstellung der Aufgabenstruktur in *JupyterLab*. Aufgabenelemente / Zellen werden hier durch blau hinterlegte Aufgaben zusammengefasst. Die Klassifizierung der Aufgabenelemente wird durch die Symbole vor den Zellen angezeigt. Die Bearbeitung des Typs findet in einem anderen Bereich statt, welcher hier nicht dargestellt ist. . . . . 41

4.3 Darstellung der Symbole, welche für die verschiedenen Aufgabenelementtypen im Plugin verwendet werden. . . . . 42

4.4 Der Screenshot zeigt die *JupyterLab*-Oberfläche, auf der die Metadaten der aktiven Zelle bzw. des *Notebook*-Dokumentes auf der rechten Seite bearbeitet werden können. . . . . 43

4.5 Ein Ausschnitt der Ansicht zur Bearbeitung der Aufgabenstruktur im implementierten Plugin. . . . . 44

4.6 Ein Ausschnitt des Plugins zur manuellen Erstellung von neuen Teilaufgaben. . . . . 44

4.7 Im unteren Bereich des Plugins wird die Klassifizierung der aktuell markierten Zelle vorgenommen. Je nach ausgewähltem Typ werden weitere Attribute zum Ausfüllen dazu angezeigt. . . . . 45

4.8 Übersicht über die im Plugin verwendeten Klassen und ihre Zusammenhänge. Es existiert maximal eine Instanz jeder Klasse. Die Typ-Definitionen, die in *Kursivschrift* dargestellt sind, stammen aus den `@jupyterlab`-Paketen. Eine Übersicht über die *React*-Komponenten wird in Abbildung 4.9 gegeben. . . . . 48

4.9 Auf der linken Seite ist die Oberfläche und auf der rechten Seite die Aufteilung in die *React*-Komponenten dargestellt. Zur besseren Übersichtlichkeit wurden nicht alle Vorkommen der Komponenten markiert. 50

5.1 Antworten der Teilnehmenden, um ihren Bezug zu Übungsblättern zu erfahren. Mehrfach-Auswahl möglich. Jede Farbe steht für eine Person. 54
5.2 Antworten auf die Frage „Wie oft arbeiten Sie mit Jupyter-Notebooks?“ Keine Mehrfach-Auswahl möglich. . . . . 54
5.3 Auswertung der berechneten *SUS-Score*s. . . . . 55
5.4 Auswertung der Aussagen zum Plugin, die von den Teilnehmenden eine hohe Zustimmung erhalten haben. . . . . 57
5.5 Auswertungen der Aussagen zum Plugin, bei denen die Teilnehmenden eher zugestimmt haben, obwohl es auch einzelne Ausreißer gab. . . . . 58
5.6 Auswertungen der Aussagen zum Plugin, bei denen die Angaben der Teilnehmenden auseinander gehen. . . . . 59
5.7 Einteilung und Zuordnung der Aufgabenelementtypen. . . . . 61
5.8 Auswertungen zum Detailgrad der Aufteilung der Aufgabenelementtypen. 62
5.9 Übersicht über die von den Teilnehmenden gewählten Typen im Beispiel-Dokument, aufgeteilt nach Zellen. Die Zellen 20 und 22 sowie 21 und 23, waren inhaltlich identisch jedoch einmal auf Deutsch und einmal auf Englisch vorhanden. . . . . 66
5.10 Grafische Darstellung eines Ausschnittes des überarbeiteten XML-Schemas für die Datenstruktur. . . . . 67
5.11 Ausschnitt aus dem überarbeiteten XML-Schema der Datenstruktur zur Darstellung der verschiedenen Aufgabenelementtypen mit ihren Attributen. Der `HintType` steht für die Optionen „constraint“, „explanation“ oder „example“. Der Typ `QStyle` steht für die Optionen „multiple-choice“, „write-code“, „explain-code“, „draw“ und „parsons-problem“. . 69

# Tabellenverzeichnis

2.1 Darstellung der Parallelen bei der Klassifizierung von Aufgaben nach verschiedenen Skill-Leveln. . . . . 19

3.1 Ergebnis der Analyse der *Notebook*-Dokumente. Anzahl Vorkommen je Kategorie gruppiert nach den Kursen der untersuchten Dokumente und Durchschnitt über die Kurse. . . . . 29

6.1 Überprüfte Aufsplittungen der Teilnehmenden der Evaluation in jeweils zwei Gruppen für die Analyse der abgegebenen Antworten. . . . . 73

# Quellcodeverzeichnis

3.1 Beispielhafte Darstellung einer stark gekürzten Datenstruktur. . . . . 33

4.1 Definition des Einstiegpunktes in das Plugin. . . . . 47

# Abstract

Das Erstellen von Feedback für bearbeitete Übungsblätter von Lernenden ist in großen Kursen sehr zeitintensiv. Im Bereich der Programmierung wird Feedback deswegen oft durch automatische Tests erzeugt. Diese können jedoch nicht auf spezifische Probleme der abgegebenen Antwort eingehen oder Fehler erklären. Lernende lernen jedoch besser, wenn sie die gemachten Fehler verstehen. Um automatisch hilfreiches Feedback zu generieren, sollen *Large Language Models* (LLMs) genutzt werden. Ein Problem bei LLMs ist, dass große Prompts zu langen Rechenzeiten führen und irrelevante Informationen ein LLM vom Thema abbringen können. Deswegen wurde eine Datenstruktur für Übungsblätter erarbeitet, welche die Erstellung der Prompts für LLMs unterstützt.

Bisherige Datenstrukturen für Übungsaufgaben sind meist für einen plattformübergreifenden Austausch entwickelt worden. Auch eine Klassifizierung von Aufgaben wird dort immer auf ganze Aufgaben bezogen. In dieser Arbeit werden Aufgaben in kleinere Elemente unterteilt, welche mit verschiedenen Typen klassifiziert werden. Dafür wurden Übungsblätter, welche in Form von *Jupyter-Notebook*-Dokumenten vorliegen, auf die unterschiedlichen vorhandenen Typen hin analysiert. Anschließend wurde mit diesen Typen und den Ergebnissen aus bisherigen Studien eine Datenstruktur entworfen. Diese wurde evaluiert und daraufhin die Datenstruktur verbessert. Für die Evaluation wurde ein Plugin für *JupyterLab* implementiert, mit dem eine Klassifizierung von Übungsblättern in Form von *Jupyter-Notebook*-Dokumenten durchgeführt werden kann.

Die entworfene Datenstruktur speichert ein Übungsblatt in Form einer XML-Datei. Eine Aufgabe wird dabei durch das XML-Element `task` dargestellt. Eine Aufgabe speichert Informationen zu Dateien, welche für eine Bearbeitung benötigt werden, sowie Abhängigkeiten zu vorherigen Teilaufgaben. Eine Abhängigkeit besteht, wenn für die Bearbeitung der Aufgabe Informationen aus einer vorherigen Aufgabenbeschreibung oder die Lösung einer vorherigen Aufgabe erforderlich sind. Zusätzlich enthält eine Aufgabe den sichtbaren Inhalt in einem `structure`-Element. Dabei werden Teilaufgaben als weiter verschachtelte `task`-Elemente repräsentiert. Die einzelnen Aufgabenelementtypen werden ebenfalls in der `structure` gespeichert. Es existieren die Typen `question`, `hint`, `answer`, `solution`, `info`, `init`, `selftests` und `grader`. Jeder Typ entspricht einem XML-Tag und kann in einem `structure`-Element beliebig oft verwendet werden. Die Typen `question`, `hint`, `solution` und `grader` können durch weitere Attribute noch genauer spezifiziert werden. Für den Typ `question` kann angegeben werden, ob es sich um eine geschlossene Frage handelt und welchem Fragetyp (Multiple-Choice, Write code, ...) sie entspricht.

Mit der entwickelten Datenstruktur können nun Übungsblätter klassifiziert werden. Das entwickelte Plugin unterstützt dabei die Klassifizierung von *Jupyter-Notebook*-Dokumenten. Anschließende Studien, welche die klassifizierten Übungsblätter zur Generierung von Feedback verwenden, sind sinnvoll. Diese Studien könnten dann evaluieren, inwiefern durch die Datenstruktur ausführlicheres Feedback generiert werden kann, welches die Lernenden beim Lernen unterstützt.

# Kapitel 1 Einleitung

## 1.1 Motivation

Zur Vertiefung von Lehrinhalten werden in der Lehre Aufgaben eingesetzt, die Lernende bearbeiten sollen oder können. Um den Lernenden eine Rückmeldung geben zu können, haben diese meist die Möglichkeit, ihre Bearbeitungen einzureichen. Um die Lernenden in ihrem Lernprozess zu fördern, ist es entscheidend, ihnen hilfreiches Feedback zu geben.

*Bisherige Varianten zur Erstellung von Feedback*

Die Erstellung von Feedback ist sehr zeitaufwendig für die Lehrenden, wenn dies manuell geschehen soll, da meist die Abgaben von vielen Lernenden korrigiert werden müssen. Alternativ können gerade bei Programmieraufgaben zum Beispiel Unit-Tests geschrieben werden, mit denen eine automatische Korrektur erstellt wird. Deren Vorteil ist, dass die Übungsblätter dann für alle Lernende neutral gleich bewertet werden. Ein Nachteil besteht darin, dass sie aufwendig bei der Erstellung sind und darauf geachtet werden muss, dass alle möglichen Antworten abgedeckt sind. Außerdem können sie nur ausgeführt werden, wenn die Abgabe der Lernenden kompiliert. Ansonsten erhalten die Lernenden als Feedback die Fehlerbeschreibungen des Compilers. Dies erschwert es insbesondere Lernenden, die noch nicht so viel Erfahrung haben, ihre Fehler zu verstehen und ihre Abgabe zu verbessern [47].

Im letzten Jahr ist das Thema *Large Language Models* (LLMs) stark in den Fokus gerückt, und es liegt nahe, dies auch bei der Feedback-Generierung zu nutzen [11]. Es gibt bereits mehrere Studien, die sich mit der Nutzung von LLMs zur Generierung von Feedback beschäftigt haben [53, 57, 17, 42, 1].

*Probleme bei der Feedback-Generierung mit LLMs*

Das Feedback, welches von LLMs generiert wird, erfüllt derzeit noch nicht zuverlässig genug die Anforderungen, die an ein gutes Feedback gestellt werden oder erzeugen im schlimmsten Fall sogar falsches Feedback. Unter anderem erfinden LLMs gerne Fehler, welche gar nicht vorhanden sind [17]. Außerdem kommt es auch vor, dass sie direkt die Lösung verraten, obwohl sie nur Hinweise geben sollen, wie eine abgegebene Lösung verbessert werden kann [17].

Des Weiteren werden in den Forschungsarbeiten zur Feedback-Generierung mit LLMs oft sehr einfach strukturierte Aufgaben genutzt, welche aus einer einzelnen Aufgabenstellung bestehen und eine Antwort dazu erwarten. In der Realität werden aber oft komplexere Übungsblätter zum Lernen bereitgestellt, welche u.a. verschachtelte Aufgabenstellungen beinhalten oder zusätzliche Hinweise für die Lernenden zur Bearbeitung mitgeben.

*Probleme von LLMs bei längeren Inputs*

Wenn nun diese komplexeren Übungsblätter in ein LLM hineingegeben werden, dann gibt es dabei noch weitere Probleme zu beachten. Generierte Texte von LLMs schweifen oft vom Thema ab, wenn andere zusätzliche Informationen mit angegeben werden, die für die Anfrage aber nicht relevant sind [63, 42]. Das bedeutet, dass andere Aufgaben eines Übungsblattes das generierte Feedback für eine einzelne Aufgabe verschlechtern.

Außerdem bedeuten längere Inputs auch mehr Inputtokens, was zu einer höheren Rechenlast führt, weshalb es oft ein Limit bei der Anzahl der Tokens gibt [53]. Wenn für die Ausführung des LLMs ein externer Service gebucht wird, werden dort die Kosten meist nach Tokens berechnet, sodass längere Inputs auch zu höheren Kosten führen [52]. Als alternative Vorgehensweise kann die Ausführung des LLM auf einem eigenen Server in Erwägung gezogen werden. Allerdings kann dies bei einer hohen Anzahl an Lernenden zu sehr langen Rechenzeiten führen. Um diesem Umstand entgegenzuwirken, besteht die Möglichkeit, die Anzahl der Server zu erhöhen, was jedoch mit einem Anstieg der Kosten verbunden ist.

## 1.2 Zielsetzung

*Eine Datenstruktur als Lösungsansatz*

Die im vorherigen Abschnitt genannten Probleme bei der Feedback-Generierung mit LLMs sollen in dieser Arbeit adressiert werden. Ziel dieser Arbeit ist es eine Datenstruktur zu entwerfen, in welcher komplexe Übungsblätter dargestellt werden können, die aus verschiedenen Teilaufgaben und zusätzlichen Angaben bestehen können. Im Rahmen dieser Arbeit wird der Fokus auf digitale Übungsblätter im Bereich der Programmierung gelegt.

Es wird untersucht, wie die automatische Generierung von Feedbacks durch eine geeignete Klassifizierung dieser Aufgaben unterstützt werden kann. Dabei soll eine Aufgabe in einzelne Elemente unterteilt werden, wobei jedes Element einer Klasse zugeordnet wird.

Diese Datenstruktur soll die Erstellung von Prompts für LLMs unterstützen, sodass die Prompts keine weiteren irrelevanten Informationen von anderen Aufgaben enthalten und deutlich kürzer werden.

*Forschungsfragen*

Dabei werden die folgenden Forschungsfragen behandelt:

- RQ1 Wie kann die automatische Generierung von Feedback durch zusätzlichen Kontext unterstützt werden?
- RQ2 Welche Anforderungen muss eine Datenstruktur erfüllen, die mit Kontext versehene Aufgaben darstellen kann?
  - RQ2.1 Welche Aufgabenelemente müssen von einer Datenstruktur unterstützt werden?
  - RQ2.2 Welche zusätzlichen, relevanten Eigenschaften können Aufgaben noch haben, die über eine Klassifizierung von Aufgabenelementen hinaus geht?
- RQ3 Wie kann die Erstellung der Datenstruktur aus einem existierenden Übungsblatt unterstützt werden?

Für die Überprüfung der Forschungsfragen, wird eine Erweiterung für die *JupyterLab*-Umgebung implementiert. Eine Einführung in *JupyterLab* und *Jupyter-Notebook*-Dokumente ist im folgenden Abschnitt zu den Grundlagen (siehe Unterabschnitt 1.3.1) gegeben. *JupyterLab* als Arbeitsumgebung für *Jupyter-Notebook*-Dokumente eignet sich gut für die praktische Erprobung der Datenstruktur. Dies ist darauf zurückzuführen, dass *Jupyter-Notebook*-Dokumente eine häufig verwendete Darstellungsform digitaler Übungsblätter für Programmierkurse sind, da sie verschiedene Medien in einem Dokument vereinen und mit einer integrierten Programmierumgebung dargestellt werden [6, 12].
*Jupyter-Notebook*-Dokumente nutzen für die Darstellung von Inhalten eine Liste von Zellen. In Bezug auf eine Datenstruktur, die einzelne Elemente einer Aufgabe klassifizieren soll, vereinfacht dies die Adressierung, da eine Aufgabe in mehrere Zellen aufgeteilt werden kann.
Ein weiterer Vorteil der *JupyterLab*-Umgebung ist, dass sie durch Erweiterungen mit neuen Funktionen ausgestattet werden kann. Dies vereinfacht die Integration der Funktionen zur Klassifizierung der Aufgabenelemente und den Export in die Datenstruktur.

## 1.3 Grundlagen

In diesem Abschnitt werden grundlegende Begriffe erklärt, die für diese Arbeit relevant sind. Außerdem werden *Jupyter-Notebook*-Dokumente und die Oberfläche *JupyterLab* vorgestellt.

### 1.3.1 Project Jupyter

*Project Jupyter* vereint verschiedene Tools und Standards für interaktive Datenwissenschaft und wissenschaftliches Rechnen [26]. Das Projekt ist aus dem Projekt *IPython* entstanden und unterstützte ursprünglich nur die Programmiersprache Python [37]. Es steuert die Entwicklung des Open-Source Standards des Datei-Formats für *Jupyter-Notebook*-Dokumente, welches zum Erstellen und Teilen von berechenbaren Dokumenten genutzt wird.

#### Jupyter-Notebook-Dokumente

*Aufbau Jupyter-Notebook-Dokumente*

Ein *Jupyter-Notebook*-Dokument besteht aus einer Liste von Zellen, die von *Jupyter-Notebook*-Dokumente werden im JSON-Format mit der Dateiendung `.ipynb` gespeichert und unterstützen verschiedene Programmiersprachen [37]. Im Standard werden sowohl Markdown-Zellen als auch Code-Zellen definiert. Markdown-Zellen sind gut für die Erstellung von erklärendem Inhalt oder Aufgabenstellungen geeignet [35], während Code-Zellen gerade im Bereich von Übungsblättern für Programmierkurse gerne als Antwortzellen genutzt werden oder für die Berechnung einer automatisierten Bewertung.

*Darstellung von Jupyter-Notebook-Dokumenten*

Zur Darstellung und Bearbeitung von *Jupyter-Notebook*-Dokumenten gibt es verschiedene Software. Die Software-Produkte, welche von *Project Jupyter* entwickelt werden, sind das *Jupyter Notebook* [23], *JupyterLab* [31] und der *Jupyter nbviewer* [22], wobei letzterer nur die Betrachtung und nicht das Bearbeiten von *Jupyter-Notebook*-Dokumenten unterstützt.

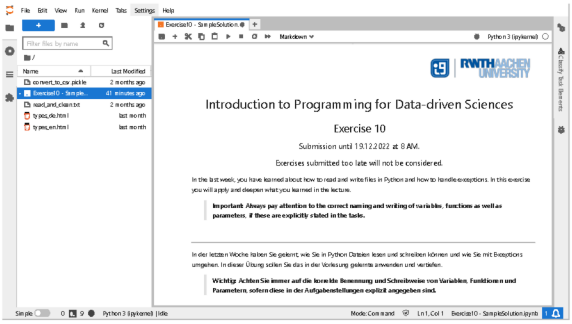



**Abbildung 1.1:** Screenshot von *JupyterLab* mit einem geöffnetem *Notebook*-Dokument

verschiedenen Typen sein können.

Da der Standard für *Jupyter-Notebook*-Dokumente ein offener Standard ist, gibt es auch weitere Drittanbieter Software, die eine Darstellung und Bearbeitung von *Jupyter-Notebook*-Dokumenten unterstützt. Beispielhaft ist hier der Editor *Visual Studio Code* genannt [24], welcher im Jahr 2021 mit 14 Millionen Nutzenden weit verbreitet war [68]. Auch *PyCharm* als Entwicklungsumgebung für Python bietet Support für *Jupyter-Notebook*-Dokumente [55].

### JupyterLab und Jupyter Notebook

*Unterschied Jupyter Notebook und JupyterLab*

*Jupyter Notebook* und *JupyterLab* sind beides webbasierte Anwendungen zur Bearbeitung und Erstellung von *Jupyter-Notebook*-Dokumenten (kurz: *Notebook*-Dokumente) [21]. *JupyterLab* unterstützt die Bearbeitung von mehreren *Notebook*-Dokumenten in verschiedenen Reitern und stellt zusätzliche Tools wie eine Konsole und eine anpassbare Oberfläche zur Verfügung [21]. *JupyterLab* ist außerdem modular aufgebaut, sodass es sich einfach durch Erweiterungen anpassen lässt [25].

### JupyterHub

*Multi-User*

Um *JupyterLab* oder auch *Jupyter Notebook* für verschiedene Nutzende zur Verfügung zu stellen, ohne dass sich die Nutzenden die Software selbst installieren müssen, kann sie mit *JupyterHub* über einen zentralen Server zur Verfügung gestellt werden. *JupyterHub* verwaltet verschiedene Nutzende und stellt den Nutzenden die jeweiligen Web-Oberflächen und Programmierumgebungen zur Verfügung. Außerdem können mit *JupyterHub* auch verschiedene Gruppen verwaltet werden, um ihnen z.B. unterschiedliche Rechte zu geben [28].

## 1.3.2 Parsons-Problem

In den verwandten Arbeiten wird das Parsons-Problem als Aufgabentyp verwendet. Daher folgt hier eine kurze Beschreibung. Das Parsons-Problem ist in der Programmierung eine Aufgabe, bei der die einzelnen Code-Zeilen vorgegeben sind. Zur Beantwortung der Aufgabe müssen die Code-Zeilen ggf. ausgewählt und anschließend in die richtige Reihenfolge gebracht werden [7]. Der Aufgabentyp stellt somit eine Alternative zum Aufgabentyp „code writing“ ([7] S. 124) dar, wobei das Parsons-Problem einfacher zu korrigieren ist und dennoch die gleichen Fertigkeiten bei den Lernenden abfragt [7].

### 1.3.3 Unterschied zwischen Aufgabenteil und Teilaufgabe

In dieser Arbeit wird für ein Aufgabenteil der Begriff *Aufgabenelement* oder *Elemente einer Aufgabe* genutzt. Dies bezeichnet meist nur einzelne Absätze innerhalb einer Aufgabe, die sich einer eigenen Kategorie zuordnen lassen, aber noch zur selben Aufgabe gehören.
Der Begriff *Teilaufgabe* steht für Unteraufgaben einer Aufgabe. Gehören zum Beispiel zur Aufgabe 1 die Aufgaben a) bis e), dann werden diese in dieser Arbeit *Teilaufgaben* der Aufgabe 1 genannt. Der Begriff *Teil(e) einer Aufgabe* wird in dieser Arbeit nicht verwendet, da bei der Benutzung nicht eindeutig unterschieden werden kann, ob eine Teilaufgabe oder ein Element einer Aufgabe gemeint ist.

## 1.4 Gliederung

Die Arbeit ist wie folgt strukturiert. Nachdem die Motivation der Arbeit, die zu beantwortenden Forschungsfragen und einige Grundlagen im Kapitel 1 Einleitung schon geklärt wurden, wird im folgenden Kapitel 2 der aktuelle Forschungsstand im Bereich der Feedback-Generierung und in der Klassifizierung von Aufgaben vorgestellt. Zuerst wird zusammengefasst welche Arten von Feedback es gibt und wie gutes Feedback aussieht (Abschnitt 2.1), um anschließend einen Überblick zu liefern, welche Ansätze es aktuell für die Feedback-Generierung gibt (Abschnitt 2.2). Als nächstes wird der aktuelle Stand der Aufgabenklassifizierung zusammengefasst (Abschnitt 2.3). Anschließend folgt eine Analyse schon bestehender Datenstrukturen für Übungsblätter in Abschnitt 2.4. Das Kapitel zu den verwandten Arbeiten wird mit einer Zusammenfassung des bisherigen Forschungsstandes abgeschlossen (Abschnitt 2.5). Dabei wird Bezug auf das Thema der Arbeit genommen und erläutert, wieso eine weitere Datenstruktur notwendig ist.
In Kapitel 3 geht es um die Erarbeitung der neuen Datenstruktur. Die Erarbeitung der Datenstruktur basiert auf einer Analyse existierender digitaler Übungsblätter und zusätzlichen Interviews mit den Erstellenden der Übungsblätter. Dafür wird als erstes die Vorgehensweise erläutert (Abschnitt 3.1) und anschließend werden die Ergebnisse der beiden Erhebungen präsentiert (Abschnitt 3.2). Aus den erhobenen Daten und den Ergebnissen aus den verwandten Arbeiten werden Anforderungen gesammelt, die von einer Datenstruktur erfüllt werden sollen (Abschnitt 3.3). Mit Hilfe dieser Anforderungen wird anschließend eine Datenstruktur entworfen (Abschnitt 3.4). Diese neu entworfene Datenstruktur wird in Abschnitt 3.5 auf ihre Nützlichkeit und ihre Beschränkungen hin diskutiert.
Im folgenden Kapitel 4 geht es um die Entwicklung eines Plugins für *JupyterLab*. Das Plugin erleichtert die Erstellung der Datenstruktur von digitalen Übungsblättern in Form von *Jupyter-Notebook*-Dokumenten. Dafür werden zuerst die Anforderungen gesammelt und ein Design-Entwurf für die Darstellung der Struktur erstellt. Im Anschluss erfolgt die Vorstellung der Struktur des implementierten Plugins sowie eine detailliertere Betrachtung ausgewählter Elemente.
In Kapitel 5 geht es um die Evaluation der Datenstruktur und die Nutzungsfreundlichkeit des entwickelten *JupyterLab*-Plugins. Für die Evaluation wurde ein Fragebogen vorbereitet, welcher in Abschnitt 5.1 vorgestellt wird. Anschließend erfolgt die Auswertung der Umfrage-Ergebnisse in Abschnitt 5.2. Dabei werden Verbesserungen an

der Datenstruktur aufgezeigt und integriert. Die daraus resultierende Datenstruktur wird in Abschnitt 5.3 präsentiert.
Im folgenden Kapitel werden die Ergebnisse der Datenstruktur diskutiert (Kapitel 6). Dabei wird zunächst auf die Methoden zur Erarbeitung der Datenstruktur eingegangen. Anschließend erfolgt eine Diskussion der Konsequenzen der Überarbeitung dieser Datenstruktur für das implementierte Plugin. Schließlich werden Limitationen der Evaluation identifiziert.
Die Ergebnisse der Arbeit werden in Kapitel 7 zusammengefasst und in Kapitel 8 wird ein Ausblick gegeben, wie die erarbeitete Datenstruktur weiter genutzt und überprüft werden kann. Zudem werden mögliche Änderungen am Plugin aufgezeigt, die zu einer Verbesserung der Nutzungsfreundlichkeit führen.

# Kapitel 2 Verwandte Arbeiten

Die Arbeit lässt sich in den Bereich der Feedback-Generierung einsortieren, da die erarbeitete Datenstruktur bei der automatischen Erstellung von Feedback unterstützen soll. Das folgende Kapitel stellt dazu vor, welche Arten von Feedback es gibt, wie gutes Feedback aussieht und welche Ansätze zur automatischen Generierung von Feedback derzeit erforscht werden.

Außerdem werden bisherige Ansätze zur Klassifizierung von Übungsaufgaben erläutert und Datenstrukturen vorgestellt, welche für den Austausch von digitalen Übungsaufgaben entwickelt wurden. Abschließend werden die Ergebnisse zusammengefasst und dabei die Grenzen der bisherigen Studien erläutert. Es wird begründet, warum eine neue Datenstruktur für die Generierung von Feedback entwickelt werden soll.

## 2.1 Was ist Feedback?

Nach Hattie und Timperley ist Feedback eine Information, die sich auf eine Leistung oder das Verständnis von Inhalten bezieht [14]. Sie kommt meist, nachdem Wissen oder eine bestimmte Verhaltensweise beigebracht wurde, und kann nur auf etwas aufbauen [14]. Um hilfreiches Feedback zu geben, muss ein Lernkontext vorhanden sein, auf den sich das Feedback bezieht [14].

*Vier Feedback-Level*

Feedback wird in deren Arbeit in vier Level eingeordnet [14]. Dabei bezieht sich Feedback, welches zum ersten Level gehört, auf die Aufgabe, wie zum Beispiel, ob eine Aufgabe richtig gelöst wurde [14]. Feedback des zweiten Levels bezieht sich auf den Prozess, welcher zum Bearbeiten einer Aufgabe genutzt wurde [14]. Das dritte Level wird als *self-regulation* bezeichnet [14]. Hiermit ist Feedback gemeint, welches das Selbstvertrauen oder die Fähigkeit zur Selbsteinschätzung fördert [14]. Als viertes Level wird Feedback beschrieben, welches sich auf die lernende Person bezieht. Es kann jedoch vorkommen, dass dieses Feedback oft auch keinen Bezug mehr zur Aufgabe hat [14].

### 2.1.1 Wie sieht gutes Feedback aus?

Gutes Feedback verbessert den Lernprozess [45]. Shute fasst zusammen, wie Feedback auf Aufgabenebene aussehen soll, damit es das Lernen verbessert [64]. Dabei werden einige Leitlinien aufgestellt. Dazu zählt, dass Feedback sich auf die Aufgabe beziehen soll und nicht auf die Lernenden [64]. Das bedeutet, dass Feedback aus dem vierten Level von Hattie und Timperley nicht als gutes Feedback angesehen wird [14].

Außerdem soll Feedback erklärend sein und nicht nur angeben, ob eine Aufgabe richtig gelöst wurde oder nicht. Gerade wenn das Feedback nur aus einer Note besteht, ist kein großer Lernzuwachs vorhanden [64]. Besser ist es, vor allem die Fragen nach dem Was, Warum und Wie zu klären [64]. Außerdem soll Feedback für die Lernenden erst sichtbar sein, sobald diese einen Lösungsversuch gemacht haben [64]. Dies passt auch zu dem Ergebnis von Hattie und Timperley, dass Feedback erst hilfreich ist, wenn auf etwas aufgebaut werden kann [14].

### 2.1.2 Verschiedene Arten von Feedback

Narciss hat eine inhaltsbezogene Klassifizierung von verschiedenen Feedback-Arten aufgestellt [48]. Dazu zählen die folgenden Typen:

Einfaches Feedback:

1. *Knowledge of performance (KP)*: Wie viele von einer Reihe von Aufgaben richtig bearbeitet wurden.

2. *Knowledge of result/response (KR)*: Ob eine einzelne Antwort richtig ist oder nicht.

3. *Knowledge of the correct result (KCR)*: Wie eine richtige Lösung aussieht, ohne auf die abgegebene Lösung einzugehen.

Ausführliches Feedback

4. *Knowledge about task constraints (KTC)*: Hinweise oder Erklärungen zu Einschränkungen der Aufgabe oder Vorgaben des Aufgabentyps.

5. *Knowledge about concepts (KC)*: Erklärung technischer Begriffe oder Beispiele auf welche Art die Aufgabe gelöst werden soll.

6. *Knowledge about mistakes (KM)*: Anzahl und Position von Fehlern, sowie Erläuterung der Fehler bzw. deren Ursprung.

7. *Knowledge about how to proceed (KH)*: Hinweise wie Fehler behoben werden können oder Fragen, welche die Lernenden zu einer richtigen Lösung leiten.

8. *Knowledge about metacognition (KMC)*: Hinweise oder Fragen, welche die Lernenden bei der Selbstreflektion ihres Lernfortschrittes unterstützen.

Die ersten drei Typen sind nicht geeignet, um die Leistung zu verbessern, für welche das Feedback gegeben worden ist [36]. Dies lässt sich auch damit begründen, da sie, wie von Shute hervorgehoben, nicht die Fragen nach dem Was, Warum oder Wie klären, sondern nur entweder eine erreichte Punktzahl (KP), ob die Lösung richtig ist (KR) oder eine richtige Lösung (KCR) angeben [64].
Die hinteren fünf Typen geben ausführlicheres Feedback und enthalten spezifischere Hinweise mit Bezug auf die bewertete Leistung und wie diese verbessert werden kann bzw. welche Stellen noch nicht richtig sind und warum [64].

*Zeitpunkt des Feedbacks*

Ein weiterer wichtiger Aspekt für Feedback ist der Zeitpunkt des Feedbacks. Dies kann während der Bearbeitung passieren, sodass die Leistung anhand des Feedbacks noch verbessert werden kann oder das Feedback wird erst nach der finalen Abgabe gegeben. Im Vergleich zu Feedback, das erst nach Abschluss der Bearbeitung gegeben wird, bietet Feedback, das während der Bearbeitung gegeben wird, eine Vielzahl von Möglichkeiten. Ein Nachteil bei Feedback, welches erst nach Abschluss der Leistung gegeben wird, ist unter anderem, dass Lernende frustriert sind, da es dann zu spät für eine Verbesserung ist [45].

Wenn Feedback nun vorab schon erhalten werden kann, führt dies bei Lernenden jedoch dazu, dass diese die Abgabe iterativ so anpassen, bis das Feedback angibt, dass die Aufgabe erledigt ist (z.B. alle Tests bestanden wurden) [45]. Dabei geht der Fokus vom eigentlichen Lernziel der Aufgabe zu einem Fokus auf die Beseitigung von Fehlern, welche durch das Feedback identifiziert wurden [45]. Wenn durch das Feedback alle Lernziele abgedeckt werden, ist das jedoch nicht problematisch.

Auf der anderen Seite kann positives Feedback, welches während der Bearbeitung gegeben wird, sobald ein Zwischenziel erreicht ist, die Lernenden auch positiv beeinflussen [19]. Somit kann verhindert werden, dass zielführende Schritte wieder rückgängig gemacht werden [19].

## 2.2 Feedback-Generierung

Im folgenden Abschnitt werden verschiedene Ansätze vorgestellt, die aktuell zur Generierung von Feedback verwendet bzw. erforscht werden. Der Fokus liegt hierbei auf der Feedback-Generierung für Aufgaben im Bereich der Programmierung.

### 2.2.1 Generelle Feedback-Generierung

*Compiler-Ausgabe*

Eine naheliegende Form für Feedback-Generierung ist dabei die Aufbereitung der Ausgaben des Compilers. Dieses Verfahren produziert jedoch kein effektives Feedback für Personen, welche das Programmieren noch neu erlernen [45].

*Automatisierte Tests*

Ähnliches gilt für die Verwendung von automatisierten Tests, die ein Programm ausführen und die Ausgabe mit der gewünschten Ausgabe vergleichen [36]. Wenn z.B. fehlgeschlagene Test-Cases als Feedback zu einer Abgabe zurückgegeben werden, hilft das meist nicht beim Finden des fehlerhaften Codes in der Abgabe, da es bei der Erstellung des Tests sehr aufwendig ist, alle möglichen Fehlerfälle abzudecken [65]. Somit wird meist nicht auf konkrete Stellen im Code hingewiesen oder ein Hinweis gegeben, wie der Fehler behoben werden kann (KH: Knowledge about how to proceed) [65].

*AST-Transformationen*

Eine weitere Variante zur Feedback-Generierung, um bei Programmieraufgaben auf Fehler hinzuweisen, ist es die Abgabe mit einer Musterlösung zu vergleichen. In der Arbeit von Suzuki et al. wird das Tool *Refazer* [58] dafür verwendet [67]. *Refazer* kann die Unterschiede zwischen zwei Programmen als Transformationen in einem `abstract syntax tree` angeben, welche dann in natürliche Sprache übersetzt werden können [67].

### 2.2.2 Feedback-Generierung mit LLMs

Auch für die Generierung von Feedback mit *Large Language Models* (LLMs) gibt es einige Ansätze, die untersucht worden sind. Viele Studien fokussieren sich aber nicht nur auf die Korrektur von abgegebenen Lösungen, sondern wollen die Lernenden schon während der Bearbeitung durch hilfreiches Feedback unterstützen.

*Fehlerspezifische Prompts*

In der Studie von Pankiewicz und Baker werden drei verschiedene Arten von Prompts getestet, je nachdem welches Problem die Lernenden gerade hatten [53]. Um das jeweilige Problem der Lernenden zu identifizieren, wurde versucht die Abgabe der Lernenden auszuführen und mit Unit-Tests zu überprüfen. OpenAIs GPT-3.5-Model wurde als LLM für die Auswertung verwendet.

Wenn nun der abgegebene Code nicht kompiliert werden konnte, dann wurde dies mit dem ersten Prompt an das LLM gesendet, welcher auch die Fehlermeldung des Compilers enthielt. Konnte der Code kompiliert werden, aber bei der Ausführung ist ein Fehler geworfen worden, wurde der zweite Prompt verwendet, welcher zusätzlich die geworfene Fehlermeldung enthielt. Wenn keine Fehler aufgetreten sind, jedoch einer der Unit-Tests fehlgeschlagen ist, dann wurden Informationen zu dem Unit-Test, wie Eingabevariablen und erwartete Ausgabe, in den dritten Prompt mitgegeben [53]. Jeder so generierte Prompt wurde zusätzlich mit der Aufgabenstellung und dem abgegebenen Code versehen. Im letzten Fall, wenn alle Unit-Tests bestanden wurden, musste kein erklärendes Feedback generiert werden und es wurde dementsprechend auch kein Prompt benötigt [53].

Eine Analyse der Aufteilung in die drei Prompts wurde nicht durchgeführt. Jedoch wurde mit einer Kontrollgruppe überprüft, ob das generierte Feedback hilft, dass mehr Lernende die Aufgaben bestehen. Dafür hatte eine Gruppe über neun Wochen für 38 von 46 Aufgaben der Studie Zugriff auf generiertes Feedback von einem LLM, während in der Kontrollgruppe kein Zugriff auf das zusätzliche Feedback bestand. Beide Gruppen hatten Zugriff auf ein einfaches Feedback von der Plattform. Für die Abgaben, für welche die Test-Gruppe auch keinen Zugriff auf das LLM hatte, hat sich herausgestellt, dass in der Kontrollgruppe mehr Prozent der Lernenden eine Abgabe erfolgreich in den ersten sechs Versuchen abgegeben haben als in der Test-Gruppe. Es wird vermutet, dass Lernende aus der Test-Gruppe sich auf das generierte Feedback verlassen haben und deswegen mehr Versuche benötigten [53].

*Next-Step Hint Generation*

In der Studie zur „Next-Step Hint Generation“ von Roest et al. wird der Fokus daraufgelegt, wie Lernenden während der Bearbeitung geholfen werden kann, die Aufgaben richtig zu bearbeiten [57]. Der Prompt wird in einer strukturierten Form erstellt und an OpenAIs GPT-3.5-Turbo-Model gesendet. Das bedeutet, dass abwechselnd jeweils eine Zwischenüberschrift und der zugehörige Inhalt zu einem Prompt zusammengesetzt werden. Die Zwischenüberschriften werden durch Zeilen wie „`--Description--`“, „`--Student Input--`“ oder „`--Instruction--`“ ([57] S. 5) angegeben. Jeder Prompt beinhaltete mindestens die Antwort der Lernenden und eine Instruktion.

Es wurden zwölf verschiedene Kombinationen ausgewertet. Vier Kombinationen wurden aus dem Vorhandensein von Aufgabenbeschreibung und Musterlösung gebildet,

welche dann mit je einer von drei verschiedenen Formulierungen für die Instruktion kombiniert wurden. Prompts, welche die Wörter „student“ ([57] S. 10) und „hint“ ([57] S. 10) enthielten, produzierten effektivere und informativere Hinweise. Prompts mit dem Wort „explain“ ([57] S. 6) führten zu längeren Ausgaben [57].

Roest et al. bewertet die Antwort des LLMs ebenfalls als effektiver, wenn nur die Aufgabenbeschreibung verwendet wurde und keine Musterlösung mit angegeben wurde [57]. Sobald eine Musterlösung mit angegeben wurde, wurde den Lernenden geraten, die eingereichte Antwort mit der Musterlösung zu vergleichen [57]. Außerdem wurden dann auch keine alternativen Ansätze, welche die Aufgabe auf eine andere Weise lösen würden als richtig erkannt. Die Lernenden erhielten stattdessen Anweisungen, wie sie ihre Lösung korrigieren können, um ihr Ergebnis der Musterlösung anzunähern, anstatt den alternativen Lösungsweg weiterzuführen [57].

Hellas et al. untersuchte ebenfalls die Auswirkung von verschiedenen Elementen in einem Prompt, welche auch durch Zwischenüberschriften miteinander zu einem finalen Prompt verbunden wurden [17]. Eines der Elemente war die Option, dass die Lernenden bei Anfragen an das LLM einen eigenen Hilfetext verfassen konnten [17]. Insgesamt standen die folgenden Elemente zur Erstellung des Prompts zur Auswahl:

1. Die Aufgabenstellung
2. Initialer Code, sofern vorhanden
3. Antwort der Lernenden
4. Hilfetext, der von den Lernenden verfasst werden konnte
5. Musterlösung
6. Ein zusätzlicher, vorgegebener Absatz, der den Kontext erklärt und nach Vorschlägen fragt

Sie stellten ebenfalls fest, dass eine mitgegebene Musterlösung oft einfach in der Antwort wiederholt wurde, anstatt ein hilfreiches Feedback zu geben. Es wurde auch häufig die Musterlösung erklärt [17]. Darüber hinaus stellte sich auch heraus, dass die Hilfetexte der Lernenden das Feedback nicht sonderlich stark beeinflussten. Ebenso schien es nicht relevant, ob trivialer, initialer Code enthalten war, wie ein leerer Funktionsrumpf [17].

*CodeHelp*

Auch bei dem Tool *CodeHelp*, welches von Liffiton et al. entwickelt wurde, hatten die Lernenden die Möglichkeit selbstständig Anfragen an ein LLM zu stellen und diese mit einer Beschreibung des Problems bzw. einer Fragestellung zu versehen [42]. Außerdem bestand ein Prompt aus der verwendeten Programmiersprache, einem von den Lernenden ausgewählter Code-Ausschnitt und einer optionalen Fehlermeldung, welche auch von den Lernenden angegeben wurde.

Anschließend wurden mit den Daten mehrere Anfragen an ein LLM ausgeführt. Die erste Anfrage an das LLM überprüft, ob genügend Informationen angegeben wurden, um die Anfrage zur Feedback-Generierung zu beantworten. Um ein besseres Ergebnis zu erhalten, wird für den Prompt die *Chain-of-Thought*-Technik angewandt, bei welcher das LLM angewiesen wird, erst zu begründen, bevor eine Entscheidung getroffen wird. Falls das LLM angibt, dass Informationen fehlen, wurde dies den Lernenden als Warnung angezeigt [42].

Unabhängig davon wurde der Prompt zur Feedback-Generierung zweimal ausgeführt und nach einer Bewertung (ohne LLM) die bessere Antwort ausgewählt [42]. Eine Antwort wurde als schlechter angesehen, wenn sie Code-Blöcke oder Begriffe aus einer Liste an Wörtern enthielt, welche vorher von der Lehrperson angegeben wurden. Damit soll verhindert werden, dass Konzepte in der Antwort des LLMs genutzt werden, welche noch nicht im Kurs beigebracht wurden. Falls die bessere Antwort noch Code enthält, wurde erneut ein LLM angefragt, um den Code zu entfernen und ggf. Erklärungen, die sich darauf beziehen, anzupassen [42]. Dies sollte verhindern, dass komplette Lösungen als Antwort zurückgegeben werden, da dies den Lernprozess nicht fördert [42].

*Genauigkeit von GPT-3*

Balse et al. untersuchten in ihrer Studie die Antworten von Lernenden in einer Prüfung zum Thema Python-Programmierung, wie zuverlässig GPT-3 diese auf Korrektheit überprüft oder verbessert [1]. Der Fokus lag hierbei auf drei Prompts:

1. Korrektheit: „Respond with incorrect if the code is incorrect. Respond with correct if the code is correct.“ ([1] S. 294)

2. Fehler benennen: „If the code is incorrect, explain what is wrong. If the code is correct, respond with correct.“ ([1] S. 294)

3. Verbesserung: „If the code is incorrect, provide suggestions to make it right. If the code is correct, suggest improvements.“ ([1] S. 294)

Pro Abgabe wurden die drei Prompts an die API von OpenAI gesendet, welche jeweils um die Aufgabenstellung und den Code der Abgabe ergänzt wurden [1].

Das Studienergebnis zeigt, dass im Durchschnitt für 71 % der Abgaben die Korrektheit von GPT-3 richtig angegeben wurde, bei 63 % die Fehler richtig benannt wurden und bei 62 % richtige Vorschläge zur Verbesserung erzeugt wurden. Mit der dritten Prompt-Variante zur Verbesserung der Abgabe erhielten je nach überprüfter Aufgabe zwischen 32 % und 93 % der Abgaben zutreffende, verbessernde Hinweise [1]. Dies ist ein sehr großer Bereich und nicht wirklich zuverlässig.

*Länge der Eingabe*

Ein weiteres Problem bei der Verwendung von LLMs durch eine API ist, dass diese oft ein Limit bei der Anzahl der Tokens aufweist, weil längere Eingaben auch rechenintensiver werden [53]. Dieses Problem besteht auch, wenn ein LLM lokal betrieben wird. Längere Eingaben begünstigen zusätzlich, dass ein LLM den Fokus verliert oder sich auf irrelevante Informationen fokussiert [63, 42].

### 2.2.3 Zusammenfassung

Feedback-Generierung ohne LLMs ist in der Lage Kompilierfehler im Code zu finden, oder fehlgeschlagene Tests zu melden, welches beides in den meisten Fällen zu *Knowledge of result/response* zählt oder viel Aufwand bei der Gestaltung der Tests benötigt. Eine Korrektur in Richtung Musterlösung ist ebenfalls möglich, aber auch hier werden konkrete Schritte genannt und nicht nur Tipps gegeben, anhand derer die Lernenden selbst neu überlegen müssen. Dies kann zu Feedback der Art *Knowledge about how to proceed* gezählt werden.

*Musterlösungen*

Bei der Feedback-Generierung mit LLMs liegt der Fokus mehr auf der Generierung von ausführlichem Feedback. Ein häufiges Problem besteht darin, dass LLMs beim Vorhandensein von Musterlösungen im Prompt sehr stark darauf fokussiert sind, diese mit der Version der Lernenden zu vergleichen oder die Musterlösung als Feedback zu verwenden [17, 57, 42].

## 2.3 Klassifizierung von Aufgaben

Um die Möglichkeiten zur Strukturierung von Prompts zu erleichtern, sollen die Aufgaben eines Übungsblattes so gespeichert werden, dass die einzelnen Elemente einer Aufgabe unterschiedlich klassifiziert werden können.

### 2.3.1 Arten der Klassifizierung

Es werden hier vor allem Arbeiten betrachtet, die sich mit der Klassifizierung von Aufgaben im Bereich der Programmierung beschäftigen. Oft geht es in diesen Arbeiten um die Klassifizierung ganzer Aufgaben anstelle der einzelnen Elemente einer Aufgabe. Die meisten Klassifizierungen dienen der Bestimmung der Schwierigkeit einer Aufgabe. In den Arbeiten wurde nach verschiedenen Kriterien klassifiziert, die im Folgenden zusammengefasst werden.

#### Art der Fragestellung

*Format der Frage*

Ruf et al. klassifiziert die Aufgaben anhand des Formats der angegebenen Information und des Formats der gewünschten Antwort [59]. Als Format wird jeweils zwischen reinen Textaufgaben, Aufgaben mit vorgegebenem Code und Aufgaben mit einem Diagramm als Eingabe unterschieden [59]. Bei der Betrachtung des Formats der Frage können auch weitere Aspekte zur Klassifizierung genutzt werden. Hierbei können Aufgabenstellungen danach unterschieden werden, ob diese in eine Geschichte verpackt werden oder ob es sich um rein technische Fragen handelt [15].

*Textaufgabe*

#### Art der Antwort

*Geschlossene Fragen*

Eine häufig genutzte Klassifizierung basiert darauf, ob die Aufgabenstellung eine geschlossene Frage beinhaltet oder nicht [15, 62, 40]. Der Term *geschlossene Frage* bzw. als Gegenstück *offene Frage* betrifft eher die Antwort anstatt der Frage und bedeutet, dass es genau eine gültige Antwort gibt und keine anderen Antworten/Varianten als richtig gewertet werden können [62, 40]. Hazzan et al. fassen die Definition noch enger und beschreiben, dass eine geschlossene Frage mit einer Liste von Antworten kommt, aus der nur noch eine ausgewählt werden muss [15].

Die von Hazzan et al. durchgeführte Klassifizierung unterteilt die Aufgabenstellungen in drei Varianten [16]. Dabei wird zunächst unterschieden, ob die Frage allgemein eine geschlossene Frage ist oder ob sie eine offene Frage ist. Die dritte Variante stellt eine normalerweise offene Frage dar, die aber so angepasst werden kann, dass nur eine Antwort möglich ist [15]. Die Arbeit von Le und Pinkwart verfeinert die Aufteilung in drei andere Varianten mit Bezug zu Programmieraufgaben [40]. Anders als bei Hazzan et al. gibt es keine Variante für normalerweise nicht geschlossene Fragen. Die folgenden Varianten werden dort verwendet [40]:

- Aufgabenstellungen, die nur eine mögliche Lösung erlauben
- Aufgabenstellungen, die mehrere Lösungen erlauben, welche aber auf derselben Idee basieren und sich nur in der Implementierung unterscheiden
- Aufgabenstellungen, die mehrere Lösungen erlauben und auch unterschiedliche Herangehensweisen

Der Fall, dass eine Lösung auf derselben Idee basiert, jedoch unterschiedlich umgesetzt wurde, wird auch in der Arbeit von Sheard et al. diskutiert. Dort wurde entschieden solche Fragen als offene Fragen zu klassifizieren [62].

*Spielraum zur Beantwortung der Frage*

Eine weniger strikte Analyse in Bezug darauf, ob eine Fragestellung eine geschlossene Frage ist oder nicht, wird von Sheard et al. in einer späteren Arbeit vorgenommen [61]. Hierbei werden die Aufgabenstellungen danach klassifiziert, je nachdem wie konkret dort vorgegeben ist, wie eine Aufgabe zu beantworten ist. Alle Aufgaben wurden immer von zwei Personen mit einem der Werte *niedrig*, *mittel* oder *hoch* für diese Kategorie versehen. Eine Zuweisung des Wertes *hoch* wurde zum Beispiel vorgenommen, wenn in der Aufgabenstellung genau beschrieben wurde, dass eine gesuchte Funktion ein Array von Integer-Werten als Parameter annehmen und die Summe aller Werte zurückgeben soll. Es wird nicht untersucht, ob Aufgabenstellungen eine geschlossene Antwort erwarten [61]. Allerdings lassen weniger explizite Aufgabenstellungen, welche mit dem Wert *niedrig* klassifiziert wurden, mehr Spielraum bei der Beantwortung der Frage zu.

*Format der Antwort*

Eine Klassifizierung der Aufgaben anhand der Art der gesuchten Antwort kann nach Ruf et al. ebenfalls nach den gleichen Formaten (Text, Code, Diagramm) geschehen, wie auch das Format der Fragestellung [59]. Petersen et al. und Sheard et al. erweitern diese drei Typen, jedoch jeweils unterschiedlich stark ausgeprägt [54, 62]. Durch die Einführung beider Arbeiten des Formats *Multiple-Choice* hat dies natürlich Einfluss auch auf die Einteilung, ob es sich um eine geschlossene Frage handelt oder nicht, sodass diese Kategorien nicht komplett getrennt voneinander betrachtet werden können.
Gerade Aufgaben mit *Multiple-Choice*-Antwortmöglichkeiten bezeichnen eher geschlossene Fragen und textuelle, erklärende Antworten eher eine offene Frage. Während in der Arbeit von Petersen et al. nur eine Unterscheidung in die vier Typen „multiple choice“, „short answer“, „writing code“ und „drawing diagrams“ ([54] S. 634) vorgenommen wird, werden die Typen in der Arbeit von Sheard et al. etwas weiter

gefasst [62]. Außerdem wird dort ein weiterer Typ, das Parsons-Problem, hinzugefügt. Die folgenden fünf Typen werden verwendet: „multiple choice, short answer [...], program code, Parsons problem and graphical representation“ ([62] S. 4).

### Erforderliche Konzepte

Bei der Arbeit von Sheard et al. wird jede Aufgabe bis zu drei Konzepten zugeordnet [62]. Teilweise bedeutete die Wahl eines Konzepts, dass andere Konzepte implizit mitgemeint sind, da sie eine Voraussetzung für das Konzept sind.

Auch in der Arbeit von Petersen et al. gibt es eine Liste von Konzepten, die sich teilweise mit den Konzepten von Sheard et al. überschneiden, aber auch andere Konzepte auflisten [54, 62]. Hier gibt es keine Beschränkung bei der Anzahl der Konzepte, die zugewiesen werden können. Sie vermuten, dass Lernende bei Aufgabenstellungen, bei denen mehrere Konzepte für die Bearbeitung benötigt werden, schlechter abschneiden, da sobald ein Konzept nicht verstanden wurde, große Teile der Aufgabe nicht bearbeitet werden können. Deswegen empfehlen sie, mehr Aufgaben zu stellen, die nur ein oder zwei Konzepte erfordern, damit die Lernenden die Möglichkeit haben, ihr Verständnis in diesen Bereichen zu zeigen.

Es ist schwierig eine vollständige Liste aller möglichen Konzepte zu erstellen bzw. diese muss gegebenenfalls je nach Datensatz angepasst werden [62]. Dies könnte mit ein Grund dafür sein, dass sich eine andere Studie nicht darauf fokussiert die benötigten Konzepte zu erfassen. Für eine Aufgabenstellung werden hier statt der benötigten Konzepte nur deren Komplexität auf einer dreistufigen Skala von *niedrig* bis *hoch* erfasst [61].

### Domänen- und kulturspezifische Referenzen

In der Arbeit von Sheard et al. wird herausgearbeitet, dass Aufgabenstellungen als unterschiedlich schwer angesehen werden können, wenn diese einen kulturellen Bezug haben [62]. Dies ist zum Beispiel der Fall, wenn eine Textaufgabe sich auf eine kulturtypische Sportart bezieht und das Fachvokabular der Sportart verwendet [62]. In der ersten Arbeit wird nur klassifiziert, ob ein solcher Bezug vorliegt [62].

Eine spätere Arbeit erfasst für die Aufgaben, wie stark Wissen aus einer anderen Domäne benötigt wird [61]. Dies wird mit den Stufen „low, medium, high“ ([61] S. 147) angegeben und ggf. um die benötigte Domäne ergänzt. Dabei wird in dieser Arbeit auch allgemeiner von einer externen Domäne anstatt einer kulturellen Referenz gesprochen [61].

### Skill-Level

Für eine Klassifizierung nach benötigten allgemeinen Fertigkeiten gibt es verschiedene Aufteilungen, die sich teilweise zusammenfassen lassen.

Bower hat sich dabei an drei Wissensstufen orientiert [3, 5]. Die erste Stufe ist deklaratives Wissen, welches Wissen über Fakten beschreibt, die eine Person hat. Die zweite Stufe beschreibt Wissen mit welchen Schritten man ein bestimmtes Ziel erreicht. Diese Stufe wird prozedurales Wissen genannt. Die letzte Stufe ist konzeptionelles Wissen. Personen, die über dieses Wissen verfügen, sind in der Lage, die Gründe für eine bestimmte Vorgehensweise zu verstehen [5]. Dazu zählt zum Beispiel das Verständnis, wie die Funktionsweise einer `for`-Schleife genutzt werden kann, um eine Lösung für eine bestimmte Aufgabe zu entwerfen [3].

In der Arbeit von Bower werden zehn Fertigkeiten herausgearbeitet, welche dort als Skill-Level bezeichnet und hierarchisch aufgebaut werden [3]. In dieser Hierarchie sind Fertigkeiten, die mehr konzeptionelles Wissen erfordern, höher eingestuft als solche, die prozedurales Wissen erfordern, und diese wiederum höher als Fähigkeiten, die deklaratives Wissen erfordern [3]. Die folgenden Typen werden in der Arbeit von Bower erarbeitet [3]:

1. Declarative Tasks
2. Comprehension Tasks
3. Debugging Tasks
4. Prediction Tasks
5. Provide-an-Example Tasks
6. Provide-a-Model Tasks
7. Evaluate Tasks
8. Meet-a-Design-Specification Tasks
9. Solve-a-Problem Tasks
10. Self-Reflect Tasks

Dabei sind höher eingestufte Typen meist komplexer oder bauen auf den vorherigen Typen auf [3].
Um die benötigten Fähigkeiten für eine Aufgabe zu klassifizieren, werden in anderen Arbeiten ähnliche Typen verwendet. Die Aufteilungen aus den Arbeiten von Sheard et al. und Harland sind identisch [62, 13]. Da die Aufteilungen aus Ruf et al. und Hazzan et al. ebenfalls sehr ähnlich sind, werden die Aufteilungen gemeinsam in Tabelle 2.1 gegenübergestellt [59, 15]. Es gibt jedoch keine Sortierung der angegebenen Typen nach ihrer Komplexität.
Einige Typen beschreiben teilweise dasselbe, wie zum Beispiel „Trace code“ ([62] S. 4) von Sheard et al. und „Tracing a given solution“ ([15] S. 5) von Hazzan et al. Jedoch wird bei Sheard et al. nur der Typ „Modify code“ ([62] S. 4) präsentiert, während dieser in den anderen beiden Arbeiten weiter unterteilt ist. Typen wie „Knowledge recall“ ([62] S. 4) oder „Design program“ ([62] S. 4) kommen in den anderen beiden Arbeiten nicht vor. Eine Klassifizierung nach verschiedenen *Skill-Leveln* ist also nicht trivial und hängt stark von den Personen ab, welche die Klassifizierung vornehmen.

**Tabelle 2.1:** Darstellung der Parallelen bei der Klassifizierung von Aufgaben nach verschiedenen Skill-Leveln.

| **Sheard et al. (2011) [62] Harland (2014) [13]** | **Hazzan et al. (2011) [15]** | **Ruf et al. (2015) [59]** |
|---|---|---|
| Knowledge recall | - | - |
| Trace code | Tracing a given solution | Trace/explain the given code |
| Explain code | - | |
| Write code | Developing a solution | Write code |
| | Developing a solution that uses a given module | Write code using the given code |
| Modify code | Completing a given solution | Adjust/extend/complete the given code |
| | - | Optimize the given code |
| | Transforming a solution from one representation to another | Transform the given code |
| Debug code | - | Debug the given code |
| Design program | - | - |
| Test program | - | Test the given code |
| - | Designing a question | Specify a problem to the given code |
| - | - | Set the right preconditions to the given code |
| - | - | Draw a diagram to the given code |
| - | Code execution analysis | - |
| - | Finding the purpose of a given solution | - |
| - | Examining the correctness of a given solution | - |
| - | Manipulating instructions | - |
| - | Estimating „efficiency“ | - |
| - | „Programming“ style questions | - |

### Intellektuelle Komplexität

Harland schlägt zusätzlich zu einer Klassifizierung nach einem *Skill-Level* auch eine Klassifizierung nach der intellektuellen Komplexität vor [13]. Auch bei Sheard et al. wird eine Klassifizierung der „Intellectual complexity“ ([61] S. 147) vorgenommen, welche die gleichen Werte nutzt. Beide verwenden für die Klassifizierung die Stufen von Blooms Taxonomie (*remember, understand, apply, analyze, evaluate, create*). Gluga et al. gibt dabei eine klare Erklärung wie diese auf Programmieraufgaben angewendet werden können [10].

#### Weitere Aspekte

*Erreichbare Punkte*

In der Literatur wird durch eine Klassifizierung der Aufgaben häufig versucht, die Schwierigkeit der Aufgabe für die Lernenden zu erfassen. In bepunkteten Abgaben wird deswegen auch vorgeschlagen, die maximal erreichbaren Punkte in einer Aufgabe im Verhältnis zu den maximal erreichbaren Punkten der kompletten Abgabe für eine Klassifizierung zu verwenden [62]. Dies kann nur verwendet werden, wenn eine Abgabe aus mehreren Aufgaben besteht, wie es jedoch bei vielen Übungsblättern und Prüfungen der Fall ist. Aufgaben, mit denen viele Punkte einer Abgabe geholt werden können, wurde dort dann als schwieriger betrachtet [62].

*Codelänge Komplexität der Sprache*

Die Betrachtung der Länge des Fragestellungs-Codes aber auch des Antwort-Codes können ebenfalls zur Klassifizierung genutzt werden [61]. Außerdem wird in der Arbeit von Sheard et al. die Komplexität und Länge des Textes der Aufgabenstellung als weiteres Kriterium genutzt [61].

### 2.3.2 Kombinationen von Aufgabentypen

Hazzan et al. weist darauf hin, dass einige Aufgaben nicht eindeutig klassifiziert werden können [15]. Hierbei wird sich auf die zwölf Typen bezogen, welche in Tabelle 2.1 zu sehen sind. Als Beispiel für eine nicht klassifizierbare Aufgabe gibt er eine Aufgabe mit mehreren Teilaufgaben an. Die gesamte Teilaufgabe lässt sich nicht einem Typen zuordnen, da die Teilaufgaben zu verschiedenen Typen gehören. Jedoch können auch innerhalb einer Teilaufgabe Aspekte aus verschiedenen Typen vorkommen. Es wird folgendes Beispiel angegeben, welches zu den Typen *Manipulating instructions*, *Examining the correctness of a given solution* und *Estimating ‚efficiency'* zugeordnet wird: „Is the solution still correct if you change the loop limit in Version B to $\sqrt{n}$ instead of $n$? If it is, what is the method efficiency after the change?" ([15] S. 13)

### 2.3.3 Zusammenfassung

*Klassifizierung von Aufgabenelementen*

Die bisher vorgestellten Arbeiten haben sich bis auf die letzte von Hazzan et al. auf die Klassifizierung von ganzen Aufgaben konzentriert [15]. Eine Teilaufgabe wurde auch bei Hazzan et al. nicht in weitere Elemente unterteilt. Dies kann jedoch relevant sein, da sich eine Teilaufgabe auch nicht immer eindeutig einem Typ zuweisen lässt [15]. In dieser Arbeit geht es um eine Klassifizierung von den Elementen einer Aufgabe. Nichtsdestotrotz können einige Aspekte der Aufgabenklassifizierung, die auch für eine Datenstruktur relevant sind, übernommen werden. Dabei geht es vor allem um Klassifizierungen, die für eine Feedback-Generierung hilfreich sein können.

*Antwort- / Frage-Format*

Dazu zählt beispielsweise die Information, ob es sich um eine geschlossene Fragestellung handelt oder nicht [15, 62, 40]. Das Format der Frage ist weniger relevant für die Feedback-Generierung jedoch für die Speicherung der Daten in der Struktur, da Bilder anders gespeichert werden sollten als Texte. Das Format der Antwort ist relevanter, wenn zum Beispiel eine textuelle Beschreibung gesucht wurde, die Antwort jedoch als Code abgegeben wurde. Auch die Angabe, ob es sich um eine geschlossene Frage handelt oder nicht, bzw. ob Antwort-Variationen als richtige Antwort gewertet

*Skill-Level*

werden sollen oder nicht, ist eine hilfreiche Information für ein Feedback. Für die Skill-Level werden in den verschiedenen Arbeiten ähnliche Ansätze genutzt, jedoch

gibt es auch dort noch einige Unterschiede bei der Klassifizierung [62, 13, 15, 59, 3]. Da diese jedoch auf einer abstrakten Ebene jeweils eine Aufgabenart beschreiben, können sie für die Feedback-Generierung hilfreich sein, um Aufgaben der gleichen Art mit der gleichen Herangehensweise zu korrigieren.

*Konzepte*

Die Bestimmung der in einer Aufgabenstellung verwendeten Konzepte gestaltet sich schwierig, da zunächst eine Liste mit den zu verwendenden Konzepten erstellt werden muss. Dies wurde in verschiedenen Arbeiten versucht, jedoch gab es dabei unterschiedliche Probleme, wie zum Beispiel, dass sich Konzepte überschneiden oder sich Aufgaben nicht eindeutig einem Konzept zuordnen lassen [62, 61, 54]. Bei Aufgaben, deren Ziel es ist, ein bestimmtes Konzept zu vermitteln, ist für das Feedback von Relevanz, ob die Aufgabe mit dem Konzept gelöst wurde oder ob ein alternativer Lösungsweg gewählt wurde. Bei komplexeren Aufgaben ist diese Angabe weniger relevant für die Erzeugung von Feedback. Die Angabe, inwiefern domänenspezifisches Wissen für die Bearbeitung erforderlich ist, ist für die Erstellung eines Feedbacks ebenso irrelevant.

*Feedback-Typen*

Aus der Feedback-Klassifizierung von Narciss können die Feedback-Typen, welche sich nicht auf eine konkrete Abgabe von Lernenden beziehen, für die Erstellung einer Datenstruktur genutzt werden [48]. Diese Feedback-Typen können bereits auf Übungsblättern enthalten sein und somit ein Element einer Aufgabe darstellen.

Ein relevanter Feedback-Typ ist *Knowledge about task constraints*, zu dem Hinweise und Erklärungen von Vorgehensweisen und Anforderungen zählen [48]. Außerdem können Definitionen oder Beispiele zu einer Aufgabe angegeben werden, welche zum Feedback-Typ *Knowledge about concepts* zählen und ebenso für eine Klassifizierung von Aufgabenelementen relevant sein können.

## 2.4 Datenstrukturen für Aufgaben

Datenstrukturen, die Aufgaben repräsentieren können, werden meistens mit dem Ziel erarbeitet, dass die Aufgaben über verschiedene Plattformen hinweg ausgetauscht werden können. Teilweise wird für die Aufgabenstellung selbst keine spezifische Struktur vorgesehen, sondern sie wird als PDF oder ähnliches gespeichert [13]. Die Struktur besteht dann hauptsächlich aus den verschiedenen Faktoren, die zur Klassifizierung genutzt werden. Jedoch gibt es auch einige Strukturen, welche die Aufgabenstellungen in einer Datenstruktur speichern, welche in den folgenden Abschnitten vorgestellt werden.

### 2.4.1 Educational Modeling Language

Eine der ersten Strukturen ist die *Educational Modeling Language (EML)*. Sie wurde 2001 von Koper präsentiert [39]. Diese Datenstruktur unterstützt die Angabe von Metadaten für eine Lerneinheit, welche aus mehreren Aufgaben bestehen kann [39]. Ziel ist es, die Wiederverwendbarkeit und Kompatibilität zu gewährleisten. Später wurde sie zu *IMS Learning Design* weiterentwickelt [51].

### 2.4.2 IMS Learning Design

Die Spezifikationen zum *IMS Learning Design (IMS LD)* wurden 2003 veröffentlicht. In dieser werden Metadaten in einer von der *IMS*-Arbeitsgruppe angepassten Form von *Learning Object Metadata* (siehe nächster Abschnitt) angezeigt [8]. Der Fokus in dieser Spezifikation wurde mehr auf die Strukturierung der Inhalte gelegt. Dabei wird der Inhalt in Aktivitäten zusammengefasst und Referenzen zu den Aktivitäten können dann in einer Aktivitäten-Struktur verschachtelt werden. In der Datenstruktur wird auch anhand von Rollen beschrieben, wem die Aktivitäten-Struktur angezeigt werden soll und wann diese als abgeschlossen gilt [18]. Damit sollen die *IMD LD*s von einem Lernmanagementsystem nutzbar werden [51].

### 2.4.3 Learning Object Metadata

Der Standard zu *Learning Object Metadata (LOM)* wurde 2002 von IEEE veröffentlicht. Darin wird beschrieben, dass die Metadaten in neun Kategorien gruppiert gespeichert werden. Die Kategorien sind *General, Lifecycle, Meta-Metadata, Technical, Educational, Rights, Relation, Annotation, Classification. LOM* ist jedoch nur zur Beschreibung der Metadaten und beschreibt nicht die Verknüpfung mit Aufgaben [41].

### 2.4.4 Sharable Content Object Reference Model

*Sharable Content Object Reference Model (SCORM)* speichert verschiedene Assets, wie HTML-Dateien, Bilder oder Audios, gruppiert in Aktivitäten. Diese können dann in einer verschachtelten Struktur dargestellt werden. Für ein so entstandenes Objekt können Metadaten angegeben werden. Hier gibt es keine weitere Vorgabe, wie diese zu strukturieren sind. Die Spezifikationen zu *SCORM* empfehlen die Verwendung von *LOM* zur Speicherung von Metadaten [60].

### 2.4.5 ProFormA

Ein weiteres Format, welches auch die Aufgabenstellung in einer Datenstruktur speichert, ist *ProFormA* [66]. Dieses Format ist speziell für die Anforderungen von Programmieraufgaben entwickelt worden und kann jeweils eine Aufgabe speichern [66].

*Version 1* Die Speicherung findet in einer XML-Struktur statt. Dabei können für eine Aufgabe neben der Aufgabenstellung und der Abgabe der Lernenden auch Tests und Musterlösungen angegeben werden. Weitere benötigte Dateien können entweder direkt eingebettet werden oder sie werden referenziert und das XML mit den Dateien zusammen in eine ZIP-Datei gepackt. Zusätzlich können mit *ProFormA* die Programmiersprache und weitere Metadaten erfasst werden. Mehrere Bewertungsschemata können ebenso inkludiert werden [66].

*Version 2* Während der Fokus bei Version 1 von *ProFormA* auf der Wiederverwendbarkeit von Aufgaben liegt, wird in der zweiten Version der Fokus auf die Kommunikation zwischen Bewertungssystemen (*Gradern*) und Lernmanagementsystemen (LMS) gelegt.

Neben dem überarbeiteten Aufgabenformat aus Version 1 wird auch ein Format für die Übergabe einer Abgabe an einen *Grader* definiert. Diese beinhaltet die studentische Lösung, die Aufgabe sowie ein möglicherweise durch die Lehrkraft angepasstes Bewertungsschema. Zusätzlich wird ein Format für ein Feedback zur Abgabe definiert, welches als Antwort vom *Grader* an das LMS genutzt wird [56].

Mit Version 2 wurde eine Internationalisierung eingeführt, sodass für jedes Feld und für alle angehängten Dateien Versionen in verschiedenen Sprachen zur Verfügung gestellt werden können [56].

## 2.5 Warum ist eine neue Datenstruktur notwendig?

*ProFormA* ist der neueste Standard zur Speicherung und zum Austausch von Übungsaufgaben, welcher auch noch aktiv entwickelt wird. Er wird auch zum Austausch von Aufgaben zwischen verschiedenen Lernmanagementsystemen genutzt. Gerade der Fokus von Version 2 auf die Kommunikation mit *Gradern* ist sehr zu befürworten und legt nahe, dass dieser Standard sich als gute Grundlage für eine Datenstruktur zur Klassifizierung von Aufgabenelementen eignet.

*Widersprüchliche Variationen*

In dieser Arbeit wird eine neue Datenstruktur erarbeitet und vorgestellt. Dies erfolgt aus verschiedenen Gründen, die im Folgenden erörtert werden. In der Praxis ist *ProFormA* zwar weit verbreitet, aber je nach Fokus in der Implementation sind nicht alle Funktionen in den jeweiligen Systemen unterstützt [56]. *ProFormA* spiegelt in der Version 1 eher eine Obermenge der Funktionalitäten wider, welche in den verschiedenen Systemen genutzt werden [56]. Dadurch gibt es verschiedenen Varianten und von diesen muss laut Spezifikation nur mindestens eine jeweils implementiert werden. Diese Varianten können sich aber in ihrer Kompatibilität gegenseitig ausschließen [56].

*Klassifizierung von Aufgabenelementen*

Des Weiteren wird in dieser Arbeit eine Klassifizierung einzelner Elemente einer Aufgabe angestrebt. Im *ProFormA*-Standard wird eine Aufgabe jedoch nicht in verschiedene Elemente aufgeteilt, sondern die komplette Aufgabe im XML-Element `description` erfasst [66]. Zwar kann die Aufgabe hier mit HTML strukturiert werden [56], jedoch müsste hier der Standard angepasst werden, um eine Klassifizierung zu erfassen. Dies kann durch benutzerdefinierte Attribute geschehen, die bei einzelnen Elementen des HTML ergänzt werden müssten. Dafür müssen Aufgaben jedoch zwangsläufig als HTML gespeichert werden und andere Formate wie zum Beispiel Markdown sind nicht möglich. Alternativ wird die Klassifizierung der Struktur durch entsprechende XML-Tags in der Struktur erfasst. Das bedeutet, dass die Kind-Elemente eines `description`-Elementes die möglichen Klassen darstellen und diese dann erst die Inhalte der Aufgabe enthalten. Letzteres ist die präferierte Variante, weil dadurch die Klassifizierung in der Datenstruktur verankert ist und nicht in dem Inhalt und die bereits genannten Probleme der Speicherung im HTML vermieden werden. Hierbei geht die Kompatibilität mit dem aktuellen Standard von *ProFormA* verloren.

*Teilaufgaben*

Ein weiterer Nachteil des *ProFormA*-Standards besteht darin, dass nur einzelne Aufgaben definiert werden können und keine Teilaufgaben [66]. Es ist also nicht möglich, die Datenstruktur weiter zu verschachteln und innerhalb einer Aufgabenstruktur noch weitere (Teil-)Aufgaben zu definieren. Für eine Feedback-Generierung sind spätere Teilaufgaben meist irrelevant und auch vorherige Teilaufgaben werden nicht immer für die Bearbeitung und somit für ein Feedback benötigt. Eine Aufteilung in Teilaufgaben ist mit der bestehenden Struktur des `task`-Elementes nicht möglich.

*Aufgaben-übergreifende Hinweise*

Eine Darstellung eines kompletten Übungsblattes mit *ProFormA* ist ebenso nicht möglich und bei der Entwicklung auch nie intendiert worden, da es ursprünglich für den Austausch einzelner Übungsaufgaben konzipiert wurde [66]. Es lässt sich diskutieren, inwieweit dies von einer Datenstruktur zur Feedback-Generierung unterstützt werden sollte. Da sich bei der Analyse der *Jupyter-Notebook*-Dokumente (siehe Unterunterabschnitt 3.2.2) herausgestellt hat, dass es Angaben oder Hinweise geben kann, die aufgabenübergreifend für ein ganzes Übungsblatt gelten, wird jedoch eine Datenstruktur bevorzugt, welche das gesamte Übungsblatt darstellen kann.

*Integration nach Auswertung*

Auf Grund der genannten Schwierigkeiten einer Integration der Klassifizierung von Aufgabenelementen in den Standard *ProFormA*, wird vorerst auf eine Anpassung des Standards verzichtet. Nachdem die Nützlichkeit der Klassifizierung in einer größeren Studie festgestellt wurde, sollte langfristig über eine geeignete Integration in den Standard von *ProFormA* nachgedacht werden. Beim Austausch einzelner Fragen mittels *ProFormA* sollten Hinweise, die für das gesamte Übungsblatt gelten, auch bei den betreffenden Fragen erfasst werden. Eine Erfassung von Teilaufgaben in dem *ProFormA*-Standard kann auch generell für die Darstellung und Feedback-Erstellung mit bisherigen Systemen von Vorteil sein.

# Kapitel 3 Erarbeitung der Datenstruktur

Dieses Kapitel beschäftigt sich mit der Entwicklung der Datenstruktur. Dazu wurden digitale Übungsblätter aus verschiedenen Kursen der RWTH in Form von *Jupyter-Notebook*-Dokumenten analysiert und verglichen. Zusätzlich wurden Interviews mit den Lehrenden geführt, welche bei der Erstellung der Übungsblätter beteiligt waren. Die Vorgehensweise dafür wird im Abschnitt 3.1 erläutert. Anschließend werden die Ergebnisse der Interviews und der Analyse der *Notebook*-Dokumente präsentiert. Daraus werden Anforderungen an eine Datenstruktur formuliert, welche in Abschnitt 3.3 aufgelistet werden. Dabei werden auch relevante Aspekte aus den verwandten Arbeiten mit eingearbeitet. Im Anschluss daran wird ein erster Entwurf der Datenstruktur vorgestellt. In Abschnitt 3.5 werden getroffene Entscheidungen für den Entwurf der Datenstruktur diskutiert.

## 3.1 Methodik

In den geführten Interviews lag der Fokus darauf zu erfahren, wie derzeit in den Kursen Feedback gegeben wird, während bei der Analyse der *Notebook*-Dokumente das Erarbeiten möglicher Typen für die Elemente einer Aufgabe im Fokus stand. Hierbei wurde auch direkt die Häufigkeit der verschiedenen Typen erfasst.

### 3.1.1 Interviews mit Erstellenden

Die Interviews wurden online über *Zoom*[1] durchgeführt. Insgesamt wurden drei Personen aus zwei verschiedenen Kursen mündlich befragt. Dabei wurde am Anfang kurz in das Thema eingeführt und anschließend wurden die Fragen durchgegangen. Mit den Fragen soll erörtert werden, wie die Korrektur derzeit durchgeführt wird, welche Art von Feedback gegeben wird und welche Anforderungen an ein Plugin zur Unterstützung bei der Klassifizierung gestellt werden. Der vollständige Fragebogen befindet sich im Anhang A.

[1] https://zoom.us

### 3.1.2 Analyse der Jupyter-Notebook-Dokumente

Für die Analyse der digitalen Übungsblätter durften die *Jupyter-Notebook*-Dokumente aus vier verschiedenen Kursen verwendet werden. Aus lizenzrechtlichen Gründen dürfen diese jedoch nicht veröffentlicht werden. Wenn mehrere Varianten zur Verfügung gestellt wurden, wurden die *Notebook*-Dokumente benutzt, welche die Musterlösung enthielten, wenn diese vorhanden waren. Die *Notebook*-Dokumente für die Lernenden waren in der Regel gleich aufgebaut, jedoch ohne die Musterlösungszellen. Stattdessen waren die Dokumente in der Regel mit leeren Zellen versehen. Für die Analyse wurden die verschiedenen Kategorien der Elemente einer Aufgabe sowie deren Häufigkeiten erfasst.

*Aufteilung nach Aufgabenelementen*

Die Klassifizierung der Aufgabenelemente orientiert sich größtenteils an den schon vorhandenen Zellen. Die Inhalte einer Zelle gehörten meist auch zur gleichen Klassifizierungskategorie. Wenn eine Zelle zu mehr als einer Kategorie zugeordnet werden konnte, wurde die Zelle für die Zuordnung so aufgeteilt, dass jeder Teil eindeutig einer Kategorie zugeordnet werden konnte. Auch wenn zwei oder mehr Elemente des gleichen Typs in einer Zelle zusammengefasst waren, wurde dies separat gezählt. Dies kam zum Beispiel häufiger bei der Musterlösung und Alternativlösungen vor, welche in der gleichen Zelle gezeigt wurden.

*Lösung / Antwortfeld*

Eine Ausnahme bildeten hierbei die Zellen, in denen in den Lösungsdokumenten die Musterlösungen erfasst wurden. Solche Zellen wurden sowohl der Kategorie „Antwortfeld“ als auch der Kategorie „Musterlösung“ zugeordnet. Es wurde angenommen, dass die Zellen bei Ausgabe als Antwortfelder für die Lernenden gedacht sind.

Die einzelnen Kategorien wurden während der Analyse erarbeitet, sodass für eine Zelle zuerst geprüft wurde, ob diese sich bereits in eine bestehende Kategorie einsortieren lässt. Falls dies nicht der Fall war, wurde eine neue Kategorie zur Liste hinzugefügt. Weitere Besonderheiten einzelner Übungsblätter wurden als Notiz zu dem jeweiligen Blatt erfasst.

*Allgemeine Infos zum Blatt*

Die Analyse der Übungsblätter wurde in verschiedene Kategorien aufgeteilt. Zur allgemeinen Struktur der Übungsblätter eines Kurses wurde erfasst, wie weit die Aufgaben auf einem Blatt verschachtelt sein können und ein Beispiel notiert, um auch die Art der Angabe wiederzugeben. Außerdem wurde festgehalten, ob in demselben Dokument mehrere verschiedene Sprachen verwendet wurden. Anschließend wurden allgemeine Kategorien für Elemente erfasst, welche auf den Übungsblättern vorkommen jedoch nicht an eine Aufgabe gebunden sind. Meist wurden Zellen dieser Kategorien am Anfang des Übungsblattes und vor der ersten Aufgabe verwendet. Lehrstoff, der auf dem Übungsblatt erklärt oder wiederholt wurde, wurde in einer eigenen Kategorie erfasst, welche ebenfalls nicht zu einer Aufgabe gehört.

*Aufgabenspezifische Kategorien*

Anschließend wurden alle Kategorien erfasst, die in den jeweiligen Aufgaben aufgetreten sind. Dazu zählt zum Beispiel die Kategorie „Beispiel(code) / Lösungsformatierung“. In dieser Kategorie wurden zum Beispiel vorhandene Code-Fragmente in

der Aufgabe erfasst, welche eine leicht andere Funktionalität als die gesuchte Antwort aufweisen und, welche bei der Strukturierung oder Formatierung des zu schreibenden Codes helfen sollen. Mit „Teil einer Antwort / Start der Antwort“ sind Code-Fragmente oder Funktionsrümpfe gemeint, welche angepasst oder ergänzt werden müssen. Eine vollständige Liste der erfassten Kategorien ist in Tabelle 3.1 zu sehen. Zusätzlich zu der Kategorie, die den Elementen einer Aufgabe zugewiesen werden konnte, wurde ebenfalls erfasst, ob (eingebettete) Bilder in der Aufgabe verwendet werden oder andere externe Dateien für eine Bearbeitung erforderlich sind.

*Feedback-bezogene Kategorien*

Im letzten Abschnitt der Tabelle wurden die Kategorien erfasst, welche einen Bezug zur Feedback Erstellung haben. Hierzu zählen Tests, die von den Lernenden vor ihrer Abgabe ausgeführt werden können, um ihre Lösung zu überprüfen, aber auch Zellen, welche für eine automatische Korrektur mit Hilfe von Tools wie *nbgrader*[2] benötigt werden. Die *Notebook*-Dokumente für die Veranstaltung *Datenbanken und Informationssysteme* speichern die Ergebnisse der jeweiligen Aufgaben dafür in ihrer eigenen Datenstruktur, der Code hierfür wird in zusätzlichen Zellen innerhalb der *Notebook*-Dokumente gespeichert.

## 3.2 Ergebnisse

Im Folgenden werden die gesammelten Ergebnisse der Interviews sowie der analysierten *Notebook*-Dokumente separat vorgestellt. Während die Interviews eine persönliche Perspektive auf die aktuelle Herangehensweise zur Korrektur von Übungsblättern geben, wird durch die Analyse der *Notebook*-Dokumente ein Einblick in die Verwendung verschiedener Kategorien für die Aufgabenelemente gegeben.

### 3.2.1 Interviews mit den Erstellenden der Übungsblätter

*Übersicht zu den befragten Kursen*

Die betreuten Kurse haben zwischen 300 und 800 Teilnehmende. Während ein Kurs sich auf die Erlernung der Programmiersprache Python für Studierende, die kein oder nur wenig Informatik in ihrem Kurs haben konzentriert, wird der andere Kurs hauptsächlich für Studierende im Studiengang Bachelor Informatik angeboten. In diesem Kurs wird Python als Mittel zum Zweck eingesetzt, um die Antworten der Studierenden direkt auswerten zu können. Weitere Programmiersprachen wie SQL, SPARQL und Gremlin werden in dem Kurs neu beigebracht.

*Aktueller Ablauf für die Korrektur*

In dem Interview wurde erörtert, wie die aktuelle Korrektur der Übungsblätter abläuft. Beide Kurse unterstützen eine vollautomatische Korrektur der Übungsblätter, welche nur in Ausnahmefällen manuell unterstützt werden müssen. Während in dem einen Kurs die Übungsblätter über einen Zeitraum von zwei Jahren mit Tests zur automatischen Korrektur versehen wurden, dauerte dies bei dem Python-Einführungskurs ca. einen Tag pro Übungsblatt.

In beiden Kursen kam es vor, dass richtige Antworten nicht als solche akzeptiert wurden, da sie von den automatischen Tests nicht erkannt wurden oder aufgrund unpräzise formulierter Aufgabenstellungen entstanden sind.

[2] `https://nbgrader.readthedocs.io`

*Feedback-qualität*

In weiteren Fragen wurde auf die Art des bisher erzeugten Feedbacks eingegangen. Hierbei wurde Bezug auf einige der Klassen von Narciss genommen [48]. Dabei stellte sich heraus, dass in beiden Kursen Feedback der Typen *Knowledge of performance*, *Knowledge of result/response* und *Knowledge of the correct result* vorhanden ist / erzeugt werden kann.

*Ausführliches Feedback*

Soweit es möglich ist, wird im automatischen Feedback auch angegeben, wo der Fehler bei der Bearbeitung ist (*Knowledge about mistakes*). Dies wird zum Beispiel durch Feedbacknachrichten bei `assert`-Statements in den Tests realisiert. Feedback des Typs *Knowledge about how to proceed* wird in dem einen Kurs nur teilweise oder gar nicht angegeben. Im Python-Einführungskurs werden nur Vermutungen zur Korrektur von Fehlern als Feedback gegeben, die aufgrund der Erfahrung von früheren Durchführungen eine der größten Fehlerquellen bei der Aufgabe waren. Es wird allerdings nicht auf die konkrete Antwort der Studierenden Bezug genommen. Eine solche Möglichkeit wurde jedoch als sehr wünschenswert angegeben.

*Anforderungen an das Plugin*

Da das Plugin für *JupyterLab* die Erstellung der Datenstruktur für *Jupyter-Notebook*-Dokumente erleichtern soll, wurde nach dem Zeitaufwand gefragt, den sie zu investieren bereit wären. Als akzeptable Dauer für die Klassifizierung pro Übungsblatt wurden sehr verschiedene Angaben gemacht: 15 bis maximal 30 Minuten, ein bis zwei Stunden und ein bis zwei Tage.

Die Befragten beider Kurse gaben an, dass es angedacht ist die Bearbeitung der *Notebook*-Dokumente über den *JupyterHub*-Server der Hochschule anzubieten. Zusätzlich wird den Studierenden ebenfalls angeboten, dass sie die *Notebook*-Dokumente lokal in ihrer eigenen Umgebung bearbeiten und ausführen können. Dies kann dazu führen, dass in den *Notebook*-Dokumenten wichtige Informationen wie Zell-IDs oder Zell-Platzhalter für die Tests verloren gehen. Dazu wurde im Interview angegeben, dass dies in 10 von 250 Fällen (4 %) bzw. im anderen Kurs in geschätzten 5 % bis 10 % der Fälle vorkommt.

### 3.2.2 Jupyter-Notebook-Dokumente

In Tabelle 3.1 sind die Ergebnisse der Analyse der *Notebook*-Dokumente in den Spalten gruppiert nach den Kursen zu sehen. In der letzten Spalte sind die kumulierten Ergebnisse über die Kurse hinweg. Vor der Berechnung des Mittelwertes wurden die Werte auf das Maximum pro Kurs normiert, um die unterschiedliche Anzahl der Elemente und Aufgaben der *Notebook*-Dokumente auszugleichen.

*Feedback in den Kursen*

Es ist wichtig zu erwähnen, dass die Kurse *Einführung in die Programmierung für datenbasierte Wissenschaften* (P4DS) und *Datenbanken und Informationssysteme* (DBIS) die Aufgaben der Studierenden automatisch korrigieren und bewerten, während die Studierenden in den anderen Kursen keine Möglichkeit auf eine automatische Korrektur ihrer Abgabe haben. Im Kurs *Text Mining* werden offenere Aufgaben gestellt, welche in einigen Punkten verschiedene Gestaltungsmöglichkeiten lassen, sodass hier nur ein manuelles Feedback angeboten wird.

**Tabelle 3.1:** Ergebnis der Analyse der *Notebook*-Dokumente. Anzahl Vorkommen je Kategorie gruppiert nach den Kursen der untersuchten Dokumente und Durchschnitt über die Kurse.

| | P4DS[a] Übungsnotebooks (WS22) Sample Solution | DBIS[b] (SS23) Source Dokumente | Text Mining (SS23) Project | Informatik im Maschinenbau (SS23) Vorgerechnete Übungsaufgaben | Informatik im Maschinenbau (SS23) Zusätzliche Übungsaufgaben | ∑ |
|---|---|---|---|---|---|---|
| Verschachtelungstiefe | 2.a) | 0.1 a) | 1.a) | f) 1)/ a) -/ a) 5. | a) | |
| Verschiedene Sprachen auf einem Blatt | x | – | – | – | – | |
| **Allgemein zum Blatt** | | | | | | |
| Titel/Abgabedatum des Blattes (Metadaten von Lehrenden) | – | x | x | x | x | |
| Informationen/Hinweise zum gesamten Blatt | 12 | 3 | 3 | 0 | 0 | 7,2 % |
| Initialisierung (z.B. pkg install) | 0[c] | 5 | 0[c] | 3 | 2 | 1,8 % |
| Metadaten (z.B. Matrikelnummer von Studierenden auszufüllen) | 0 | x | x | 0 | 0 | 40,0 % |
| **Info/Read only** | | | | | | |
| Erklärung/Tutorial von (neuem) Stoff | 3 | 4 | 0 | 9 | 12 | 5,0 % |
| **Aufgaben spezifisch** | | | | | | |
| Aufgabenbeschreibung / Einschränkung | 18 | 32 | 0 | 28 | 33 | 20,5 % |
| Beispiel(code) / Lösungsformatierung | 1 | 28 | 0 | 6 | 6 | 7,6 % |
| Hinweise zur Aufgabe | 22 | 23 | 3 | 13 | 7 | 16,6 % |
| Tipps bei Problemen / Falls vorher nicht gelöst | 6 | 0 | 0 | 0 | 1 | 1,6 % |
| Aufgabenstellung / Fragestellung | 76 | 97 | 16 | 133 | 113 | 96,7 % |
| Teil einer Antwort / Start der Antwort | 15 | 103 | 5 | 24 | 8 | 34,8 % |
| Antwortfeld | 1[d] | 103 | 16 | 71 | 75 | 64,2 % |
| Bilder zur Erklärung | 11 | 0 | 0 | 0 | 0 | 2,6 % |
| Externe Dateien (*.csv, . . . ) | 4 | 0 | 3 | 2 | 0 | 5,0 % |
| **Feedback** | | | | | | |
| Prüfcode / Selbsttests / Beschreibung des gewünschten Outputs | 32 | 40 | 0 | 15 | 22 | 21,4 % |
| Grader (z.B. *nbgrader*) | 0 | 101 | 0 | 0 | 0 | 19,6 % |
| Musterlösung | 85 | 103 | 0 | 100 | 83 | 69,7 % |
| Ignorieren | 1 | 1 | 0 | 1 | 2 | 0,9 % |
| Ergebnissicherung (DBIS[b] spezifisch) | 0 | 70 | 0 | 0 | 0 | 13,6 % |

[a] Einführung in die Programmierung für datenbasierte Wissenschaften
[b] Datenbanken und Informationssysteme
[c] Kein Code nur Hinweise im Text auf benötigte Pakete
[d] Es wurden nur die Lösungsdokumente untersucht

*Unterschiede zwischen den Kursen*

Es lässt sich feststellen, dass je nach Kurs die Kategorien von Aufgabenelementen unterschiedlich stark verwendet wurden. Gerade bei den Übungsblättern der Veranstaltung *DBIS* wurde häufig Beispielcode zur Verfügung gestellt, welcher einen Anhaltspunkt für den Aufbau der Antwort der Aufgabe liefern sollte. Außerdem wurde in diesem Kurs auch häufig ein Aufgabenstart vorgegeben. Das bedeutet, dass in dem Antwortfeld oft schon Code vorhanden war, welcher ergänzt werden sollte. Dies können Funktionsrümpfe, Variablennamen oder ähnliches sein.
Während in den Übungsblättern zur *DBIS*-Veranstaltung keine externen Dateien für die Bearbeitung verwendet wurden, kam dies aber in ein paar Dokumenten aller anderen Veranstaltungen vor. Bilder, die inhaltlich für Aufgaben verwendet wurden und in die Übungsblätter integriert waren, kamen jedoch nur bei der Veranstaltung *P4DS* vor. Ebenso gab es bei P4DS insgesamt nur sechs Tipps bei Problemen oder falls die (Teil-)Aufgabe vorher nicht gelöst wurde. Bei den Zusatzübungen für *Informatik im Maschinenbau* war es sogar nur ein Tipp.

*Gemeinsamkeiten*

Neben der offensichtlichen Gemeinsamkeit, dass jede Aufgabe eine Aufgabenstellung enthält, war auch ca. ein Fünftel der Aufgaben mit weiteren Einschränkungen versehen. Auch ein Startpunkt für eine Antwort wurde oft vorgegeben. Selbst wenn man die Werte des DBIS-Kurses aus der Rechnung entfernt, hatten im Schnitt noch ca. 15 % der Aufgaben einen Teilcode vorgegeben, welcher schon im Antwortfeld enthalten war.
Im Bereich des Feedbacks gab es für alle Kurse außer Text Mining zu fast jeder Aufgabe mindestens eine Musterlösung, teilweise sogar weitere alternative Lösungswege. Auch wurde bei diesen Kursen bei ca. 21,4 % der Aufgaben eine Überprüfungsmöglichkeit für die Studierenden mitgeliefert, sodass diese zumindest Teile ihrer Antwort vor einer Abgabe überprüfen konnten.
Vereinzelt wurden in den *Notebook*-Dokumenten auch Zellen als Trennelemente verwendet, entweder in Form von leeren Zellen als eine Art Abstandshalter oder auch mit horizontalen Linien zur Trennung von Aufgaben. Dies wurde in der Kategorie „Ignorieren“ erfasst, da diese Zellen keine inhaltliche Relevanz haben.

### Besonderheiten einzelner Notebook-Dokumente

In diesem Abschnitt werden einzelne Vorkommnisse in Übungsblättern erwähnt, welche nicht in die Übersichtstabelle mit aufgenommen wurden.

*Multiple-Choice*

In den Übungsblättern der DBIS-Veranstaltung gibt es zwei Aufgaben, welche nicht nach zu produzierendem Code fragen, sondern vom Typ *Multiple-Choice* sind. Zur automatischen Auswertung innerhalb der *Jupyter-Notebook*-Dokumente, müssen die ausgewählten Antwortmöglichkeiten jedoch auch in einer Variable gespeichert werden.

*Aufgabenstellung im Output*

Bei einer Aufgabe wird die Aufgabenstellung nicht wie üblich in einer Zelle vom Markdown-Typ dargestellt, sondern durch das Ausführen von Code erzeugt. Dabei werden bei der Ausführung Variablen im Text eingesetzt und die Aufgabenstellung befindet sich im Output der Zelle und nicht in dem Inhalt. Gerade wenn das *Notebook*-Dokument zurückgesetzt worden ist, ist die eigentliche Aufgabenstellung dann somit nicht in dem *Jupyter-Notebook*-Dokument als Ganzes abgespeichert vorhanden.

*Referenz zu anderem Übungsblatt*

Bei den Übungsblättern von *Text Mining* wird auf Blatt 3 erwähnt, dass man hier den Code von Übungsblatt 1 verwenden kann. Dies ist jedoch eine Ausnahme bei den untersuchten Übungsblättern, sonst wird nur innerhalb eines Blattes bzw. einer Aufgabe auf andere Aufgabenteile referenziert.

*Freitextantworten*

Sowohl im Kurs *Informatik im Maschinenbau* als auch bei *Text Mining* gibt es insgesamt drei Aufgaben, bei denen eine Antwort als Freitext gesucht ist. Da dies auch die beiden Kurse sind, welche keine automatische Korrektur anbieten, gibt es hier auch keine Probleme mit regelbasierten *Gradern*. Durch die Nutzung von *Natural Language Processing* (NLP), welches zum Beispiel direkt in der Form von LLMs stattfindet, kann hier möglicherweise demnächst auch Feedback automatisch generiert werden.

*Bildantworten*

Ebenso kommt bei den Übungsblättern aus dem Kurs für die Studierenden im Bereich Maschinenbau auch eine Fragestellung vor bei der als Antwort ein Diagramm gezeichnet werden soll. Somit kommen in den analysierten Übungsblättern auch alle drei Darstellungsarten („Code“, „Text“ und „Diagramm“ ([59] S. 11)) einer Antwort vor, wie sie in der Studie von Ruf et al. herausgearbeitet wurden [59].

*Eine Antwort aus mehreren Teilaufgaben*

Eine letzte Besonderheit gibt es noch im Kurs *Informatik im Maschinenbau*, denn dort gibt es eine Aufgabe, welche aus mehreren Teilaufgaben besteht, welche Schritt für Schritt eine Lösung aufbauen. Das bedeutet, dass es hier nicht pro Teilaufgabe ein Antwortfeld gibt, sondern die Code-Änderungen der einzelnen Teilaufgaben im gleichen Antwortfeld vorgenommen werden. Dies hat für eine Feedback-Generierung den Nachteil, dass die einzelnen Antworten je nach Ergebnis nicht mehr den einzelnen Teilaufgaben einfach zuzuordnen sind.

*Abhängigkeiten von Teilaufgaben*

Ein weiterer Aspekt, dessen Erfassung nicht vorgesehen war, wurde bei der Analyse der *Notebook*-Dokumente deutlich. Es wurde ursprünglich angenommen, dass sich Teilaufgaben auf die jeweilige vorherige Teilaufgabe beziehen und Aufgaben so Stück für Stück aufgebaut werden. Im Laufe der Analyse stellte sich jedoch heraus, dass einige Teilaufgaben unabhängig von vorherigen Teilaufgaben waren. Zudem basierten Aufgaben in der Regel auf Informationen aus den jeweiligen Aufgabenstellungen, nicht jedoch auf den Lösungen. Dies ermöglichte es Lernenden, spätere Teilaufgaben zu lösen, ohne die vorherigen bearbeiten zu müssen.

*Aufgabenübergreifende Angaben*

In einigen Übungsblättern werden auch allgemeine Hinweise zur Bearbeitung der Aufgaben gegeben, welche für alle Aufgaben eines Übungsblattes gelten. Teilweise werden auch Bibliotheken initialisiert, welche für mehrere Aufgaben benötigt werden. Das bedeutet, dass Aufgaben nicht abgeschlossen für sich betrachtet werden können, sondern immer im Kontext des Übungsblattes betrachtet werden müssen.

## 3.3 Anforderung an eine Datenstruktur

Aus den Ergebnissen der Interviews und der Analyse der *Jupyter-Notebook*-Dokumente wurden die folgenden Anforderungen an eine Datenstruktur herausgearbeitet. Anforderungen an das *JupyterLab*-Plugin werden in Kapitel 4 erarbeitet.
Einige ursprünglich separat gezählte Kategorien wurden zu einem Typ zusammengefasst, da sie aus Sicht der Feedback-Generierung die gleiche Bedeutung haben. Die Kategorie „Ergebnissicherung (DBIS spezifisch)“ ist für das automatisierte bewerten durch Grader relevant. In diesen Zellen wird jedoch nur vorbereitend das Ergebnis der Lernenden abgespeichert, um anschließend durch Zellen der Kategorie „Grader“ Punkte zu vergeben. Die Aufgabenelemente, die zur Kategorie der Ergebnissicherung gehören, benötigen in der Datenstruktur jedoch keinen eigenen Typ.
Titel der Aufgaben und des Blattes sowie allgemeine Daten wie Kursname, herausgebender Lehrstuhl und weitere Metadaten sind für die Feedback-Generierung nicht von Relevanz, da sie keine wichtigen Informationen zur Bearbeitung der Aufgaben enthalten. Dazu zählen auch Elemente, die von den Studierenden z.B. mit ihrem Namen oder ihrer Matrikelnummer ausgefüllt werden müssen.

- R1 Unterstützung für Teilaufgaben
- R2 Dateien mit Aufgaben verknüpfen
- R3 Aufteilung einer Aufgabe in verschiedene Elemente mit den folgenden Typen
  - R3.1 Aufgabenstellung
  - R3.2 Aufgabenbeschreibung / Einschränkungen
  - R3.3 Hinweise zur Aufgabe
  - R3.4 Hinweise zu allen Aufgaben eines Übungsblattes
  - R3.5 Antwortfeld
  - R3.6 Angabe der Lösung
  - R3.7 Titel der Aufgabe / Meta-Informationen
  - R3.8 Initialisierungen
  - R3.9 Erklärung von (neuem) Lernstoff
  - R3.10 Beispiel-Code
  - R3.11 Code oder Beschreibungen zur eigenständigen Überprüfung der Antwort
  - R3.12 Grader-spezifische Elemente wie zum Beispiel Code zur Berechnung der erreichten Punkte
- R4 Pro Aufgabe sollen mehrere Elemente des gleichen Typs erlaubt sein
- R5 Die Reihenfolge der Elemente soll erhalten bleiben
- R6 Ein vollständiges Übungsblatt mit mehreren Aufgaben soll in einer Datenstruktur darstellbar sein
- R7 Aufeinander aufbauende (Teil-)Aufgaben sollen angegeben werden können

## 3.4 Datenstruktur - Erster Entwurf

Aus den Anforderungen des vorherigen Abschnittes wurde eine erste Datenstruktur entworfen. Die Speicherung erfolgt dabei im XML-Format. Das dazugehörige XML-Schema wird in Abbildung 3.1 und Abbildung 3.2 grafisch dargestellt. Eine Aufgabe wird dabei durch ein `task`-Element dargestellt, welches die beiden Attribute `fromPrevious` und `bullet` besitzt (siehe dazu exemplarisch Zeile 28 in Quellcode 3.1). Wenn `fromPrevious` vorhanden ist, kann es den Wert `description` oder `result` enthalten, um die Art der Abhängigkeit von vorherigen Aufgaben anzugeben. Hiermit wird die Anforderung R7 erfüllt. Das Attribut `bullet` soll den Text der Aufgabenbezeichnung enthalten, wie zum Beispiel „Aufgabe 1“ oder „b)“. Im Folgenden wird ein verkürztes Beispiel einer Datenstruktur präsentiert, welches das Verständnis erleichtern soll.

```
1  <task bullet="Exercise10 - SampleSolution.ipynb">
2    <externalFiles/>
3    <structure>
4      <info>Introduction to Programming for Data-driven Sciences [...]</info>
5      <hint>In the last week, you have learned about how to read and [...]</hint>
6      <!-- [Task 1 entfernt] -->
7      <task bullet="Task 2">
8        <externalFiles/>
9        <structure>
10         <info># Task 2 (8 points)
11 Every year it is time for St. Nicholas to go around the [...]</info>
12         <task bullet="a)">
13           <externalFiles/>
14           <structure>
15             <question qstyle="write-code">## a)
16 First you should define an exception ‘ChildNotFoundException‘ [...]</question>
17             <hint>&gt; **Note:** Use the given constant
                   ↪ ‘ERROR_MESSAGE_CHILD_NOT_FOUND‘ to construct [...]</hint>
18             <!-- [Weitere Aufgabenelemente entfernt] -->
19             <selftests>For help: / Als Hilfe:
20 Make sure that the following code works with your [...]</selftests>
21             <init>ERROR_MESSAGE_CHILD_NOT_FOUND = &amp;quot;The child {} was not
                   ↪ found in the omniscient book.&amp;quot;</init>
22             <solution preferred_solution="true">class ChildNotFoundException(
                   ↪ Exception):
23     def __init__(self, child):
24         Exception.__init__(self, &amp;quot;Child not founds&amp;quot;)
25     [...]</solution>
26           </structure>
27         </task>
28         <task bullet="b)" fromPrevious="description">
29             <!-- [Inhalt von Aufgabe 2 entfernt] -->
30         </task>
31         <!-- [Weitere Aufgaben entfernt] -->
32       </structure>
33     </task>
34   </structure>
35 </task>
```

**Quellcode 3.1:** Beispielhafte Darstellung einer stark gekürzten Datenstruktur.

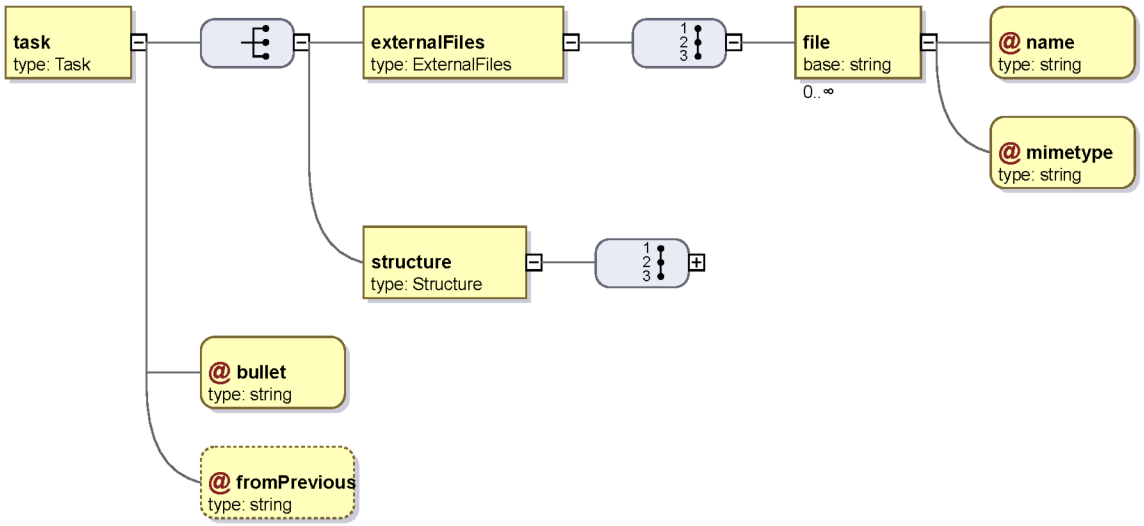



**Abbildung 3.1:** Darstellung des XML-Schemas für einen `task` im ersten Entwurf der Datenstruktur. Das `structure`-Element wird hier gekürzt dargestellt und ist in Abbildung 3.2 vollständig zu sehen.

Jedes `task`-Element besitzt zwei Kinder `externalFiles` und `structure`. Ersteres wird dazu genutzt die Dateien zu verknüpfen, welche für eine Aufgabe benötigt werden und bezieht sich damit auf die Anforderung R2. Die Dateien werden bei Text-Dateien direkt als Text eingefügt und binäre Dateien werden mit base64 in Zeichen umgewandelt, sodass diese ebenfalls inline in das XML eingefügt werden können.
Das `structure`-Element enthält die eigentlichen Texte und Code-Stücke einer Aufgabe. Um die Anforderung zur Darstellung von Teilaufgaben (R1) zu erfüllen, können als Kind-Elemente ebenfalls weitere `task`-Elemente genutzt werden. Als Wurzel-Element wird ebenfalls ein `task`-Element verwendet, um die verschiedenen Aufgaben auf einem Übungsblatt zu vereinen (R6). So lassen sich auch generelle Hinweise o.Ä., welche aufgabenübergreifend gelten sollen darstellen.
Jeder Aufgabenelement-Typ und jedes `task`-Element kann beliebig oft und in beliebiger Reihenfolge vorkommen, jedoch sollte die Reihenfolge der Inhalte identisch mit der Reihenfolge auf dem Übungsblatt sein (R4, R5).
Ein Aufgabenelement-Typ enthält als Kind direkt den Text der Aufgabe, welcher auch mit Markdown formatiert sein darf. Zusätzlich können je nach Typ noch weitere Eigenschaften in den Attributen gespeichert werden. Eine Übersicht über die einzelnen Typen wird in Abbildung 3.2 gegeben.

*Typen von Aufgabenelementen*

Die Ergebnisse der Analyse der *Notebook*-Dokumente und der verwandten Arbeiten wurden auf die folgenden Typen von Aufgabenelementen reduziert:

- **`question`** - Dieser Absatz beschreibt die Frage oder die Aufforderung an die Lernenden. Hierbei kann angegeben werden, ob es sich um eine geschlossene Fragestellung handelt oder nicht. Zusätzlich kann in dem Attribut `qstyle` der Fragetyp gespeichert werden (siehe Zeile 15 in Quellcode 3.1), dabei stehen die folgenden Werte zur Auswahl: `multiple-choice`, `write-code`, `explain-code`, `draw`, `parsons-problem` (R3.1).

- **`constraint`** - Dies ist ein Zusatz zu der gegebenen Aufgabe und gibt oft weitere Kontextinformationen oder Einschränkungen zu der Aufgabe an (R3.2).

- **`hint`** - Diese Information soll den Lernenden bei der Lösung der Aufgabe helfen, indem sie zum Beispiel bestimmte Funktionen angibt, die zu berücksichtigen sind (R3.3). Durch die Verwendung eines `task`-Elementes als Wurzelknoten lassen sich hiermit auch allgemeine Hinweise für alle Aufgaben eines Blattes erfassen (R3.4).
- **`answer`** - Hier können die Lernenden ihre Antwort eintragen. Teile der Antwort können bereits vorgegeben sein (R3.5).
- **`solution`** - Für eine Aufgabe kann es mehrere Lösungsbeispiele geben. Die Lösung, welche keine Alternative ist, soll als bevorzugte Lösung markiert werden. Hierfür wird das boolesche Attribut `preferred_solution` verwendet, wie in Zeile 22 des Quellcodes 3.1 dargestellt (R3.6).
- **`info`** - Dies beschreibt Aufgabentitel oder andere Informationen, die für die Bearbeitung der Aufgaben nicht relevant sind (R3.7).
- **`init`**- Dieser Abschnitt kann Code-Fragmente enthalten, welche die Programmierumgebung für die Aufgabe initialisieren und nicht geändert werden müssen (R3.8).
- **`explanation`** - Dies sind Informationen, die neue Konzepte erklären oder z.B. neue Formeln liefern, die zur Lösung der Aufgabe benötigt werden (R3.9).
- **`example`** - Beispiele bestehen oft aus Code, welcher der erwarteten Lösung ähnlich ist und den Lernenden bei der Formulierung ihrer eigenen Lösung helfen kann (R3.10).
- **`selftests`** - Elemente dieses Typs sollen den Lernenden helfen, ihre Lösung selbst zu überprüfen. Dies kann ein Testfall sein oder auch nur eine Beschreibung der erwarteten Ausgabe (R3.11).
- **`grader`** - Bewertungszellen können eine Punktzahl für Lernende berechnen und werden den Lernenden oft nicht zur Verfügung gestellt. Hierbei kann angegeben werden, wie viele Punkte maximal durch einen Test erreicht werden können und nach Abgabe der Antwort der Lernenden, kann hier auch direkt die erreichte Punktzahl gespeichert werden (R3.12).

## 3.5 Diskussion

In diesem Abschnitt werden wichtige Designentscheidungen der Datenstruktur diskutiert.

*XML-Format*

Die Datenstruktur wird im XML-Format gespeichert, denn XML ist ein weit verbreitetes Format, welches sich gut für die strukturierte Speicherung von Daten eignet. Hierbei können Elemente verschachtelt werden und die Reihenfolge der Elemente wird beachtet. Zusätzlich können XML-Schemas definiert werden, mit denen die Struktur der XML-Dokumente vorgegeben bzw. überprüft werden kann. Ein weiterer Punkt, der für die Verwendung von XML spricht, ist, dass dadurch eine mögliche spätere Integration in das *ProFormA*-Format erleichtert wird.

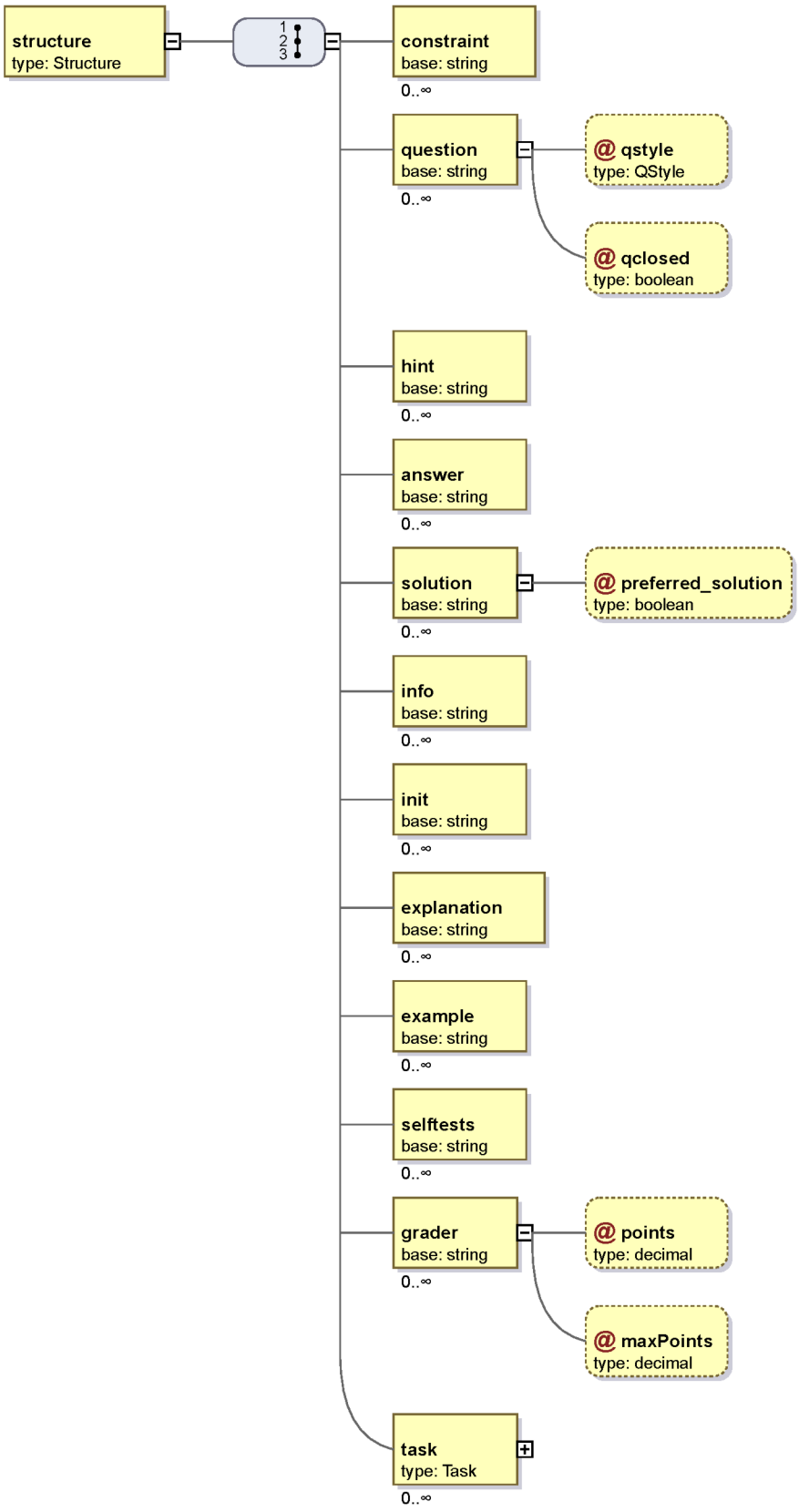



**Abbildung 3.2:** Darstellung des XML-Schemas für die Struktur der Aufgabenelemente in einem `structure`-Element im ersten Entwurf der Datenstruktur. Der Typ `QStyle` für das entsprechende Attribut bei dem `question`-Element steht für die Optionen „multiple-choice“, „write-code“, „explain-code“, „draw“ und „parsons-problem“. Das `task`-Element wird hier gekürzt dargestellt und ist in Abbildung 3.1 vollständig zu sehen.

*task-Element als Wurzel*

Für das Wurzel Element der XML-Struktur wird kein neues Element eingeführt, sondern das `task`-Element genutzt. Die Übungsblätter der Veranstaltung *Informatik im Maschinenbau* bestehen nur jeweils aus einer Aufgabe, sodass diese Entscheidung hier sinnvoll ist. Bei den anderen Übungsblättern sind jedoch mehrere Aufgaben vorhanden, welche durch die Datenstruktur zusammengefasst werden sollen. Es gelten jedoch fast die gleichen Anforderungen wie bei der Zusammenfassung von Teilaufgaben zu einer Aufgabe: Die Reihenfolge der Aufgaben muss erhalten bleiben, allgemeine Hinweise gelten für alle Aufgaben auf dem Übungsblatt, und die Aufgaben können von anderen Aufgaben auf dem Übungsblatt abhängig sein. Aus diesen Gründen erschien es sinnvoll, das `task`-Element dort ebenfalls zu verwenden.

*Dateien als base64*

Die Datenstruktur sieht vor, dass angehängte Dateien in dem XML gespeichert werden. Dies hat den Vorteil, dass eine Darstellung eines Übungsblattes in der Datenstruktur nur eine Datei umfasst. Alternativ könnte alles in eine zip-Datei wie bei *ProFormA* gebündelt werden und die angehängten Dateien würden nur referenziert. Dies bringt jedoch hauptsächlich einen Mehrwert bei der manuellen Bearbeitung der XML-Datei mit einem generischen (Text-)Editor. Eine solche Bearbeitung ist aber nicht vorgesehen und erhöht damit nur die Komplexität beim Im- und Export der Datenstruktur.

*Markdown-Unterstützung*

Bisher wird eine Formatierung mit Markdown für die Aufgabenelemente unterstützt. Da viele Aufgaben keine aufwändige Formatierung benötigen, sollte das für die meisten Aufgaben ausreichend sein. Markdown unterstützt nativ auch die Verwendung von HTML-Tags [46]. Die Verwendung von HTML kann jedoch dazu führen, dass zu viele irrelevante Informationen für die Generierung von Feedback vorhanden sind. Dies kann beispielsweise in Form von `style`-Attributen oder HTML-Tags erfolgen, die lediglich für das Layout verwendet werden, jedoch keinen Einfluss auf die Struktur haben.

*`qstyle` und `qclosed`*

Die zusätzlichen Attribute „qstyle“ und „qclosed“ wurden aus den Verwandten Arbeiten aufgegriffen und hier integriert. Der Hintergrund ist, dass je nach Fragetyp ein Feedback unterschiedlich ausfallen sollte, da die Lernenden bei geschlossenen Fragen ganz andere eingeschränkte Antwortmöglichkeiten haben. Denkbar wäre hier auch je nachdem wie die Attribute ausgefüllt sind, die jeweiligen Antworten der Lernenden inklusive der Aufgabe an unterschiedliche Systeme zur Generierung von Feedback zu geben.

*Punkte bei `grader`-Elementen*

Zu den `grader`-Elementen wird gespeichert, wie viele Punkte maximal erreicht werden können, wenn zum Beispiel der Unit-Test erfolgreich durchläuft. Sobald die Antwort der Lernenden vorhanden ist, kann dann die erreichte Punktzahl für diesen Test berechnet und ebenfalls in der Struktur gespeichert werden. Dadurch können fehlgeschlagene Tests identifiziert werden. Die fehlgeschlagenen `grader`-Elemente können bei der Generierung von Feedback insofern hilfreich sein, dass sie meist bestimmte Aspekte der Antwort überprüfen und somit bei der Feedback-Generierung einen Fokus legen können, wo ein Fehler in der Antwort der Lernenden ist.

*Evaluation*

Die hier beschriebene Datenstruktur wurde zusammen mit dem entwickelten Plugin in einer Studie evaluiert. Die Evaluation findet sich in Kapitel 5. Basierend auf der Evaluation wurden die dort erfassten Verbesserungen und Probleme in der Datenstruktur angepasst und in Abschnitt 5.3 als überarbeitete Datenstruktur präsentiert.

# Kapitel 4 Entwicklung des Plugins

Dieses Kapitel beschäftigt sich mit der Entwicklung des Plugins. Der erste Abschnitt erklärt, welchen Zweck das Plugin erfüllen soll und wer zur Zielgruppe gehört. Der nächste Abschnitt stellt die konkreten Anforderungen und eine geeignete Integration in *JupyterLab* vor, während der nachfolgende Abschnitt die Funktionen des Plugins beschreibt. Der letzte Abschnitt behandelt die Projektstruktur.

## 4.1 Zweck und Anwendungsbereich

Das *JupyterLab*-Plugin soll bei der Klassifizierung von digitalen Übungsblättern in Form von *Jupyter-Notebook*-Dokumenten unterstützen. Diese Klassifizierung muss einmalig bei der Erstellung des Übungsblattes erzeugt werden. Im Falle späterer Änderungen an den Aufgaben ist auch eine Anpassung der entsprechenden Klassifizierung erforderlich. Da die Erstellung und Änderungen des Übungsblattes in der Regel von den Lehrenden vorgenommen werden, sind diese folglich die aktiven Nutzenden des Plugins und somit die primäre Zielgruppe.

*Ablauf der Benutzung*

Ein Ablaufdiagramm zur Erstellung einer Datenstruktur für die Feedback-Generierung ist in Abbildung 4.1 dargestellt. Nach einer abgeschlossenen Klassifizierung kann das Übungsblatt an die Lernenden ausgegeben werden. Diese bearbeiten anschließend die Aufgaben und brauchen von dem Plugin nichts zu bemerken. Anschließend werden die Antworten wieder eingesammelt. Dies ist unabhängig von dem Plugin und kann zum Beispiel über bestehende Lernmanagementsysteme oder ähnliche Infrastruktur stattfinden.

Solange kein von den Lernenden genutzter Editor Informationen aus dem *Jupyter-Notebook*-Dokument löscht, ist das Dokument weiterhin klassifiziert. Eine Ausnahme bilden die Zellen, die von den Lernenden neu hinzugefügt oder durch Duplizieren entstanden sind. Da Lernende nur Antworten eingeben sollen und keine weiteren Aufgaben erstellen, ist hier eine automatische Klassifizierung der neuen Zellen vom Typ `answer` sinnvoll. Die Zuordnung der neuen Zellen zu einer Aufgabe kann automatisch über die benachbarten Zellen erfolgen.

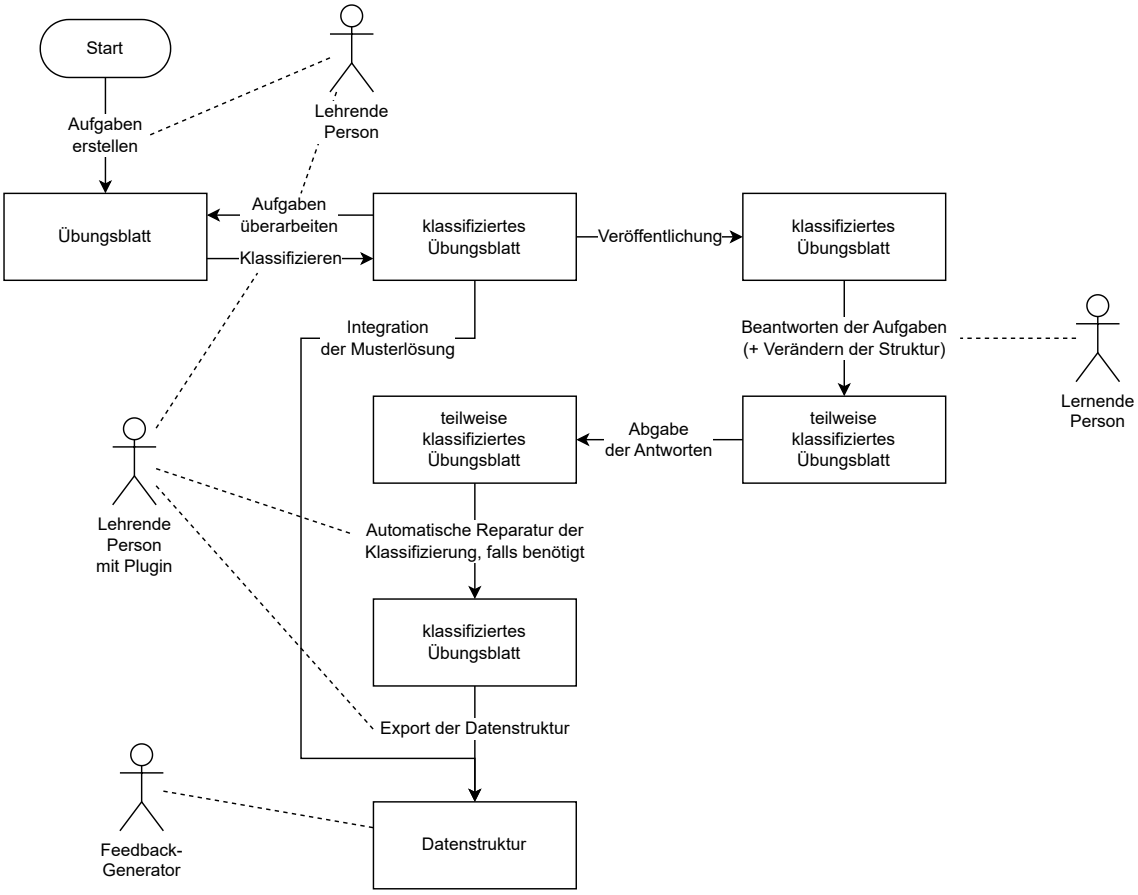



**Abbildung 4.1:** Ablaufdiagramm zur Erstellung einer Datenstruktur für die Feedback-Generierung

Aus den Übungsblättern können nun die Datenstrukturen exportiert werden, welche zur weiteren Verwendung an einen Feedback-Generator gegeben werden können. Dabei ist zu beachten, dass die Übungsblätter der Lernenden keine Musterlösungen enthalten können, da *Jupyter-Notebook*-Dokumente im JSON-Format gespeichert sind und Musterlösungen somit einfach auszulesen wären. Auf der anderen Seite enthalten die *Notebook*-Dokumente mit den Musterlösungen nicht die Antworten der Lernenden, sodass diese beiden Informationen für die Datenstruktur zusammengefügt werden müssen. Durch die Strukturierung in Aufgaben und Teilaufgaben sollte ein Erkennen zusammengehöriger Elemente jedoch zu keinen Komplikationen führen. Ebenso müssen nur die Antwortelemente vom Blatt der Lernenden zu der Datenstruktur hinzugefügt werden.

## 4.2 Entwurf

Vor der Implementierung des Plugins wurden die wichtigsten Anforderungen gesammelt und ein erstes Design für die Darstellung der Zugehörigkeit der Zellen zu den Aufgaben erstellt. Dafür wurden auch verschiedene Stellen zur Integration in *JupyterLab* getestet. Die Vor- und Nachteile der verschiedenen Optionen werden abschließend diskutiert.

### 4.2.1 Anforderungen

*Nutzungsfreundlichkeit*

Das Plugin soll die Erstellung der Datenstruktur ermöglichen (A1) und dabei möglichst nutzungsfreundlich zu verwenden sein (A2).

*Speicherung in den Metadaten*

Die Klassifizierung des Übungsblattes soll in dem *Notebook*-Dokument gespeichert werden, um Anpassungen an dem *Notebook*-Dokument auch direkt in der Datenstruktur machen zu können (A3). Für die Speicherung von zusätzlichen Informationen in einem *Notebook*-Dokument gibt es die `metadata`-Bereiche, in denen Daten im JSON-Format gespeichert werden können [49]. Diese Bereiche sind für jede Zelle und für das gesamte *Notebook*-Dokument verfügbar. Zu den Zellen soll die jeweilige Klassifizierung mit den weiteren Attributen je nach Typ gespeichert werden.

*Unterstützung für eindeutige IDs*

Die Strukturierung in Aufgaben und Teilaufgaben soll in den Metadaten des *Notebook*-Dokumentes gespeichert werden. Dafür müssen die Zellen eindeutig identifiziert werden. Seit Version 4.5 des Dateiformats für *Jupyter-Notebook*-Dokumente hat jede Zelle eine eindeutige ID [50, 38]. In den analysierten *Notebook*-Dokumenten war die niedrigste vorhandenen Version 4.4. Durch ein Speichern in Version 4.5 erhalten alle Zellen eine eindeutige ID, welche anschließend genutzt werden kann. Das Plugin soll also ein Upgrade der *Notebook*-Dokumente auf Version 4.5 unterstützen (A4).

*Datenstruktur reparieren*

Aus den Interviews lässt sich ableiten, dass die meisten *Notebook*-Dokumente von den Lernenden während der Bearbeitung nicht wesentlich in ihrer Struktur verändert werden. Die in Einzelfällen gelöschten Zellen können bei einer Zusammenführung der Antworten mit der vorher exportierten Datenstruktur ignoriert werden, solange die Strukturierung in die Aufgaben weiterhin erkennbar bleibt. Ideal ist es, wenn Zellen, welche von den Lernenden neu erstellt werden, direkt bei der Erstellung automatisch durch das Plugin als Antwortzellen klassifiziert werden können. Auch kann die Zuordnung zu den Aufgaben während der Bearbeitung durch das Plugin aktuell gehalten werden, falls zum Beispiel Zellen in der Reihenfolge verschoben werden. Dazu muss jedoch sichergestellt sein, dass dieses Plugin auch bei den Lernenden installiert ist.
Laut Interviews ist es zwar angedacht, die Lernenden die Abgabe in einem zur Verfügung gestellten *JupyterHub* bearbeiten zu lassen, jedoch kann eine lokale Bearbeitung mit anderer Software nicht ausgeschlossen werden. Deshalb ist es sinnvoll, eine nachträgliche automatische Reparatur der Datenstruktur in den *Notebook*-Dokumenten zu unterstützen (A5).

### 4.2.2 Design

In Abbildung 4.2 ist zu sehen, wie eine Strukturierung eines Übungsblattes in Aufgaben und Teilaufgaben dargestellt werden kann. Jede Zelle wird dabei durch den Anfang ihres Inhaltes repräsentiert. Zusätzlich wird die Klassifizierung einer Zelle durch ein Symbol in der jeweiligen Zeile angezeigt. Eine Aufgabe fast mehrere Aufgabenelemente bzw. Zellen und Teilaufgaben zusammen. Die Aufgaben werden durch die blau gefärbten, abgerundeten Rechtecke dargestellt und haben in diesem Entwurf zusätzlich zu ihrer Nummerierung (wie „1“, „a“, ...) noch einen Titel.
Auf den Titel wurde bei der Implementierung jedoch verzichtet, da er in der Datenstruktur für eine Feedback-Generierung nicht relevant ist und deswegen nicht vorgesehen ist. Zusätzlich ist ein Titel der Aufgabe für die Lernenden meist sichtbar in das *Notebook*-Dokument geschrieben, sodass er auch Inhalt einer Zelle ist, welche der Aufgabe zugeordnet ist.

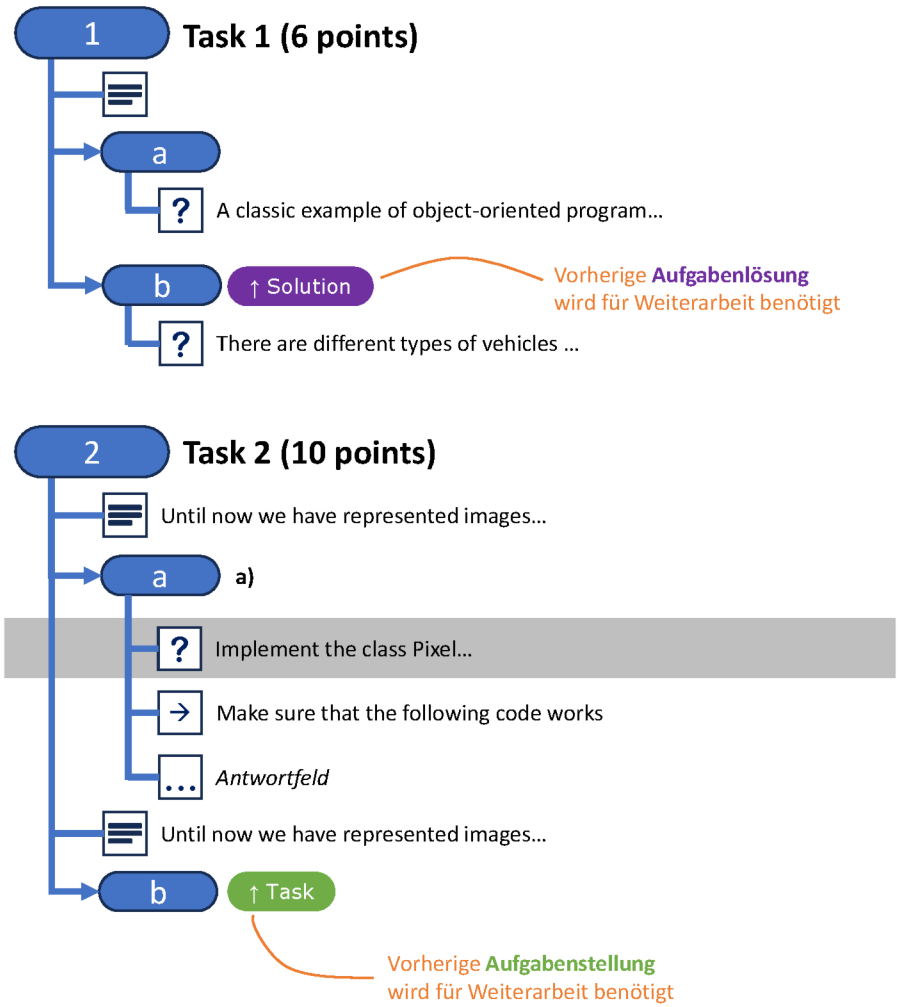



**Abbildung 4.2:** Entwurf zur Darstellung der Aufgabenstruktur in *JupyterLab*. Aufgabenelemente / Zellen werden hier durch blau hinterlegte Aufgaben zusammengefasst. Die Klassifizierung der Aufgabenelemente wird durch die Symbole vor den Zellen angezeigt. Die Bearbeitung des Typs findet in einem anderen Bereich statt, welcher hier nicht dargestellt ist.

Die Abhängigkeit von vorherigen Aufgaben ist in früheren Entwürfen noch mit Pfeilen auf die vorherige Aufgabe dargestellt worden. Dies war jedoch sehr unübersichtlich, sodass hier eine Darstellung in Form von Chips[1] stattfindet.

Die hellgrau markierte Zelle wird aktuell im *Notebook*-Dokument bearbeitet. Durch einen Klick auf eine Zelle in der Struktur soll zur dazugehörigen Zelle in dem *Jupyter-Notebook*-Dokument gesprungen werden.

Der Typ jedes Aufgabenelementes wird durch ein Symbol am Anfang der Zeile angegeben. Für die verschiedenen Typen wurden unterschiedliche Symbole entworfen. Die Symbole, welche auch im Plugin verwendet werden, sind in Abbildung 4.3 dargestellt und wurden nach den Design-Entwürfen erstellt.

[1] `https://m3.material.io/components/chips/overview`

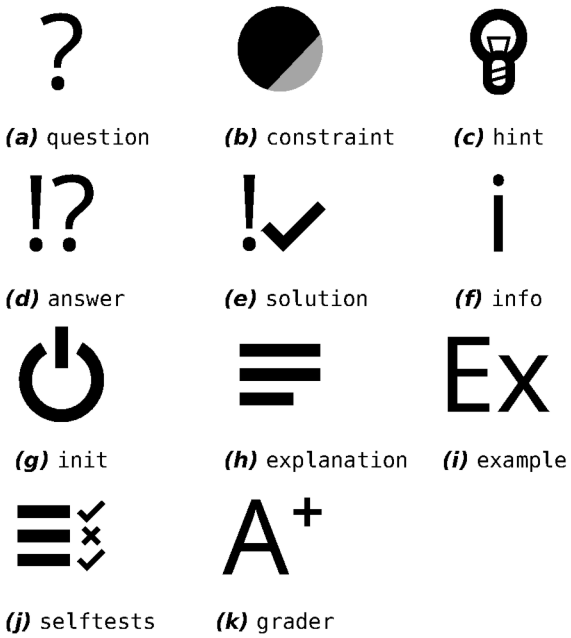



**Abbildung 4.3:** Darstellung der Symbole, welche für die verschiedenen Aufgabenelementtypen im Plugin verwendet werden.

### 4.2.3 Integration in JupyterLab

Wie in Anforderung A3 schon erwähnt, sollen die Angaben in den Metadaten des *Notebook*-Dokumentes bzw. den jeweiligen Zellen gespeichert werden. Mit *JupyterLab* können die Metadaten eines *Notebook*-Dokumentes bzw. der Zellen in der Seitenleiste direkt bearbeitet werden (siehe Abbildung 4.4). Diese Bearbeitungsmöglichkeit der Metadaten kann durch ein JSON-Schema [20] so angepasst werden, dass die Werte nicht manuell im JSON-Format geschrieben werden müssen, sondern durch geeignete HTML-Formular-Elemente unterstützt werden.

*JSON-Schema für Metadaten-Editor*

Ein einfaches Dropdown-Element, um den Typ einer Zelle anzugeben, wäre damit sehr einfach umzusetzen. Das in Abbildung 4.2 beschriebene Design lässt sich in diesen vordefinierten Metadaten-Editor auch integrieren. Hierfür muss jedoch eine eigene Komponente geschrieben werden und es reicht nicht, die Struktur per JSON-Schema vorzugeben. Die von den jeweiligen Typen abhängigen zusätzlichen Attribute lassen sich im JSON-Schema beschreiben, sodass diese nur eingeblendet werden, wenn der passende Typ ausgewählt wurde.

*Gründe gegen den Metadaten-Editor*

Eine Darstellung im Metadaten-Editor erscheint aus mehreren Gründen ungeeignet. Zum einen können die anderen Elemente aus dem Editor nicht ausgeblendet werden, sodass eine manuelle Bearbeitung der rohen Metadaten nicht ausgeschlossen werden kann. Dies kann schnell zu Metadaten führen, welche nicht dem JSON-Schema entsprechen und somit zu einer fehlerhaften Datenstruktur führen. Zum anderen geht durch die bereits vorhandenen Einträge im Editor vertikaler Platz verloren. Wenn noch weitere Plugins installiert sind, die dort Einträge hinzufügen, kann schnell viel Platz verloren gehen. Mehr vertikaler Platz verbessert jedoch die Übersichtlichkeit der Darstellung der Aufgabenstruktur, wie in Abbildung 4.2 gezeigt.

*Vorteile eigenes Panel*

Daher wurde beschlossen, die Benutzeroberfläche des Plugins in einem eigenen Panel mit einem Reiter in die Seitenleiste auszulagern. Weitere Vorteile dieser Variante

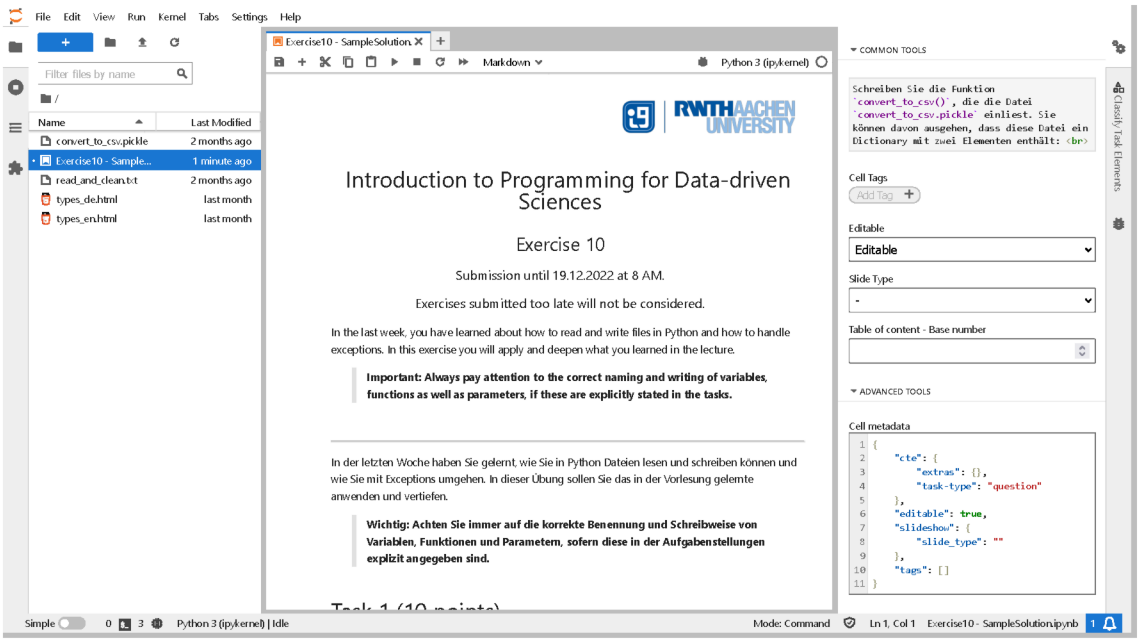



**Abbildung 4.4:** Der Screenshot zeigt die *JupyterLab*-Oberfläche, auf der die Metadaten der aktiven Zelle bzw. des *Notebook*-Dokumentes auf der rechten Seite bearbeitet werden können.

sind ein einfacheres Ausblenden des Plugins und die Lehrpersonen können selbst entscheiden, ob sie die Klassifizierung auf der linken oder rechten Seite anzeigen lassen wollen.

## 4.3 Funktionsübersicht

Der folgende Abschnitt stellt die verschiedenen Funktionen vor, die in dem Plugin implementiert sind.

### 4.3.1 Initiale Erstellung der Datenstruktur

Um die Strukturierung in Aufgabe und Teilaufgabe in einem existierenden Übungsblatt zu vereinfachen, kann diese automatisch erkannt werden. Voraussetzung dafür ist die Installation des Plugins `@jupyterlab/toc` in *JupyterLab*, welches für die Erstellung eines Inhaltsverzeichnisses anhand der vorhandenen Überschriften zuständig ist. Dabei werden sowohl mit Markdown formatierte Überschriften als auch mit HTML formatierte Überschriften in dem *Jupyter-Notebook*-Dokument erkannt. Das Plugin `@jupyterlab/toc` ist seit *JupyterLab* Version 3.0 standardmäßig integriert [34].

### 4.3.2 Bearbeitung der Struktur

Neben der Erstellung der initialen Struktur ist auch eine Bearbeitung möglich. Das entworfene Design wurde dafür um zusätzliche Symbole an der rechten Seite einer Aufgabe ergänzt, wie in Abbildung 4.5 dargestellt. Mit diesen Symbolen lassen sich die Nummerierung editieren, Dateien verknüpfen und die Aufgabe weiter einrücken, um sie als eine Teilaufgabe der übergeordneten Aufgabe zu klassifizieren. Wenn eine

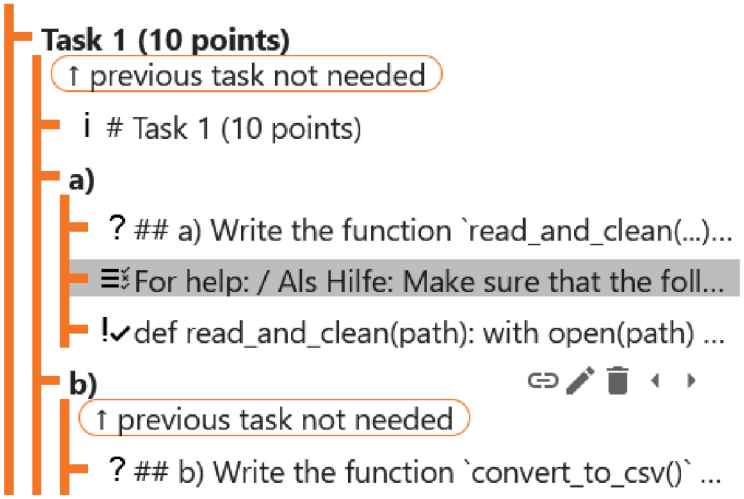



**Abbildung 4.5:** Ein Ausschnitt der Ansicht zur Bearbeitung der Aufgabenstruktur im implementierten Plugin.

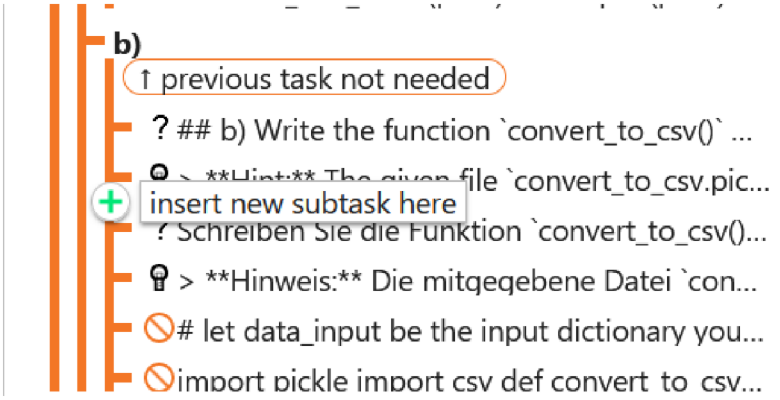



**Abbildung 4.6:** Ein Ausschnitt des Plugins zur manuellen Erstellung von neuen Teilaufgaben.

Aufgabe wieder zurückgerückt wird, dann ist sie keine Teilaufgabe mehr, sondern wird als nächste Aufgabe (auf der gleichen Ebene) klassifiziert.

Zusätzlich lässt sich noch die Gruppierung als eine Aufgabe entfernen (durch den Mülleimer symbolisiert), dabei bleiben die dazugehörigen Zellen sowohl im *Notebook*-Dokument als auch in der Datenstruktur erhalten. Um eine neue Aufgabe in der Datenstruktur zu erstellen, kann zwischen zwei Zell-Repräsentationen in der Ansicht auf „insert new subtask here“ geklickt werden, wie in Abbildung 4.6 dargestellt.

### 4.3.3 Klassifizierung der Aufgabenelemente

Für die markierte Zelle kann im unteren Bereich der Seitenleiste eine Klassifizierung vorgenommen werden. Dabei wird als erstes ein passender Typ ausgewählt. Wenn für den Typ noch weitere Attribute angegeben werden können, dann werden diese nach der Auswahl eingeblendet und können ebenfalls ausgefüllt werden. In Abbildung 4.7 ist dies für den Typ `question` dargestellt.

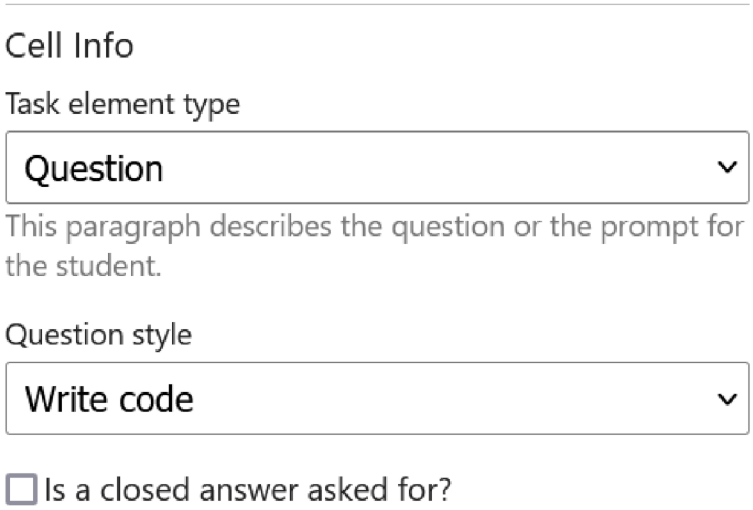



**Abbildung 4.7:** Im unteren Bereich des Plugins wird die Klassifizierung der aktuell markierten Zelle vorgenommen. Je nach ausgewähltem Typ werden weitere Attribute zum Ausfüllen dazu angezeigt.

### 4.3.4 Automatische Reparatur

Sollten die vorhandenen Zellen in dem *Notebook*-Dokument einmal nicht mit der Struktur in den Aufgaben und Teilaufgaben übereinstimmen, dann wird im Plugin ein Fehler angezeigt. Wie bereits in Anforderung A5 erläutert, ist hier eine automatische Reparatur sinnvoll. Diese kann durch einen Klick auf den entsprechenden Button ausgeführt werden.

Wenn lediglich Zellen entfernt oder hinzugefügt werden können, ist dies durch einen einfachen Vergleich der IDs möglich. Dabei würden Zellen aus dem *Notebook*-Dokument, die in der Struktur nicht vorhanden sind, an passender Stelle eingefügt werden. Zellen, welche nicht mehr in dem *Notebook*-Dokument vorhanden sind, würden einfach entfernt werden. Jedoch ist es auch möglich, Zellen in einem *Notebook*-Dokument zu verschieben, wobei sie ihre ID behalten.

*Levenshtein-Algorithmus*

Die Reparatur berechnet deswegen zuerst mit Hilfe des Levenshtein-Algorithmus die minimale Anzahl an Änderungen, welche in der Reihenfolge der Zellen in der Struktur vorgenommen werden müssen, um die gleiche Reihenfolge wie in dem *Jupyter-Notebook*-Dokument zu erhalten. Anschließend werden diese Änderungen an der bestehenden Struktur vorgenommen, sodass neue Zellen automatisch zu einer (Teil-)Aufgabe zugeordnet werden können und der Rest der Struktur erhalten bleibt.

*Reparatur der Metadaten*

Von der Reparatur ist nur die Struktur des *Notebook*-Dokumentes betroffen, da die Klassifizierung der einzelnen Zellen in die Typen in den Metadaten der jeweiligen Zellen gespeichert ist und bei einer Umsortierung bei der zugehörigen Zelle gespeichert bleibt. Sollten bei einer Abgabe alle Metadaten gelöscht worden sein, so könnten diese über eine Zuordnung über die Zell-IDs von dem ursprünglichen *Jupyter-Notebook*-Dokument wiederhergestellt werden. Dies bezieht sich jedoch schon auf das Zusammenführen der Abgabe der Lernenden mit zum Beispiel den Musterlösungen und ist nicht Bestandteil des Plugins.

### 4.3.5 Sichtbarkeit des Plugins

Der im vorherigen Abschnitt beschriebene Reparaturmechanismus ist dazu geeignet, fehlerhafte Dokumente zu korrigieren. Diese entstehen, wenn Änderungen an der Struktur vorgenommen werden, ohne dass das Plugin im Hintergrund läuft. Bei Lehrenden, die das Plugin nutzen, stimmt die Struktur somit immer mit den Zellen überein, da die Struktur bei Zelländerungen sofort vom Plugin angepasst wird. Dieser Vorteil kann auch bei den Lernenden genutzt werden, wenn diese in einer Umgebung arbeiten, in der das Plugin installiert ist. Die Lernenden sollen jedoch keine Klassifizierung der Aufgabenelemente verändern oder die Struktur anpassen und möglicherweise Aufgaben löschen können.

*JupyterHub Integration*

In den Interviews wurde gesagt, dass den Lernenden generell eine Umgebung über *JupyterHub* zur Verfügung gestellt wird. Sobald das Plugin mit einem *JupyterHub* betrieben wird, werden je nach Rollen der Benutzenden die verschiedenen Funktionen aktiviert. Die Lehrenden erhalten vollen Zugriff auf alle Funktionen, während die Lernenden von dem Plugin nichts sehen und das Plugin nur im Hintergrund die Struktur aktuell hält. Wird das Plugin nicht mit einem *JupyterHub* betrieben, dann kann der volle Umfang in den Einstellungen des Plugins aktiviert werden. Standardmäßig ist erstmal nur die Lernenden-Version aktiv.

## 4.4 Implementierung

Dieser Abschnitt stellt die Strukturierung des Source-Codes des *JupyterLab*-Plugins vor.

*Technologien*

*JupyterLab* ist eine webbasierte Oberfläche [25]. Das bedeutet, dass auch die Oberflächen der Plugins im Browser dargestellt werden und mit HTML und CSS formatiert sind. *JupyterLab* verwendet als Grundlage für die Gestaltung *Lumino* Widgets [30]. Neben den Widgets für die Oberfläche stellt das *Lumino* Package zusätzlich Layouts, Events und Datenstrukturen zur Verfügung, welche im JavaScript-Code genutzt werden können.

Das implementierte Plugin basiert auf der von *Project Jupyter* bereitgestellten Vorlage für Erweiterungen [32]. Für die Programmierung wird die Sprache *TypeScript*[2] verwendet, welche vor Auslieferung des Plugins in JavaScript übersetzt wird. Große Teile der Oberfläche des Plugins sind als *React*[3]-Komponenten geschrieben. *JupyterLab* unterstützt die Verwendung von *React* in *Lumino*-Widgets durch die Klasse `ReactWidget` aus dem Package `@jupyterlab/ui-components` [33].

Ein *JupyterLab*-Plugin wird durch ein Objekt des Typs `JupyterFrontEndPlugin` definiert. In Quellcode 4.1 ist ein Ausschnitt der Definition des Einstiegpunktes des Plugins zu sehen.

[2] `https://www.typescriptlang.org`
[3] `https://react.dev/`

```
const plugin: JupyterFrontEndPlugin<void> = {
  id: "classify_exercise_task_elements:settings_schema",
  description:
    "A JupyterLab extension to help classifying task elements in jupyter
        notebook documents (to improve feedback generation).",
  autoStart: true,
  requires: [INotebookTracker, ISettingRegistry, IDocumentManager],
  optional: [ITableOfContentsTracker, JupyterFrontEnd.IPaths],
  activate: (
    app: JupyterFrontEnd,
    notebookTracker: INotebookTracker,
    settingRegistry: ISettingRegistry,
    docManager: IDocumentManager,
    tocTracker: ITableOfContentsTracker | null,
    paths: JupyterFrontEnd.IPaths | null
  ) => {
    // Activation code of the plugin
  },
};
```

**Quellcode 4.1:** Definition des Einstiegpunktes in das Plugin.

Jedes Plugin benötigt eine eindeutige ID und eine `activate`-Funktion [27, 43]. Zusätzlich können über die Felder `description` und `autoStart` eine kurze Beschreibung des Plugins angegeben und *JupyterLab* mitgeteilt werden, dass die `activate`-Funktion direkt beim Start von *JupyterLab* auch ausgeführt werden soll.

*Zugriff auf andere Plugins*

Die meisten Funktionen von *JupyterLab* werden durch Erweiterungen zur Verfügung gestellt, wobei es einige `core plugins` gibt, welche nach der Installation von *JupyterLab* automatisch installiert und aktiviert sind [29]. Da das Plugin auch Funktionen aus anderen Plugins von *JupyterLab* benötigt, müssen die dazugehörigen Plugins bei der Definition mit angegeben werden. Dies geschieht über die beiden Felder `requires` und `optional`, je nachdem, ob Teile des Plugins auch ohne Zugriff auf die externen Funktionen nutzbar sind oder nicht. Die `activate`-Funktion bekommt nun als Parameter einen generellen Zugriff auf die Schnittstelle der *JupyterLab*-App und auf die Schnittstellen der angefragten Plugins, falls diese installiert sind.

In Abbildung 4.8 sind die implementieren Klassen und ihre Referenzen untereinander abgebildet. Die Datei `index.ts` definiert den Einstiegspunkt und enthält die in Quellcode 4.1 definierte Plugin-Definition. In der dortigen `activate`-Funktion werden die Objekte der anderen Klassen initialisiert.

### DataController

*JupyterHub-Controller*

Der `JupyterHubController` prüft, ob *JupyterLab* in einer *JupyterHub*-Umgebung ausgeführt wird. Falls dies der Fall ist, wird auch anhand der Rolle der nutzenden Person geprüft, ob sie eine Lehrperson ist. Beide Informationen können von der Klasse abgefragt werden.

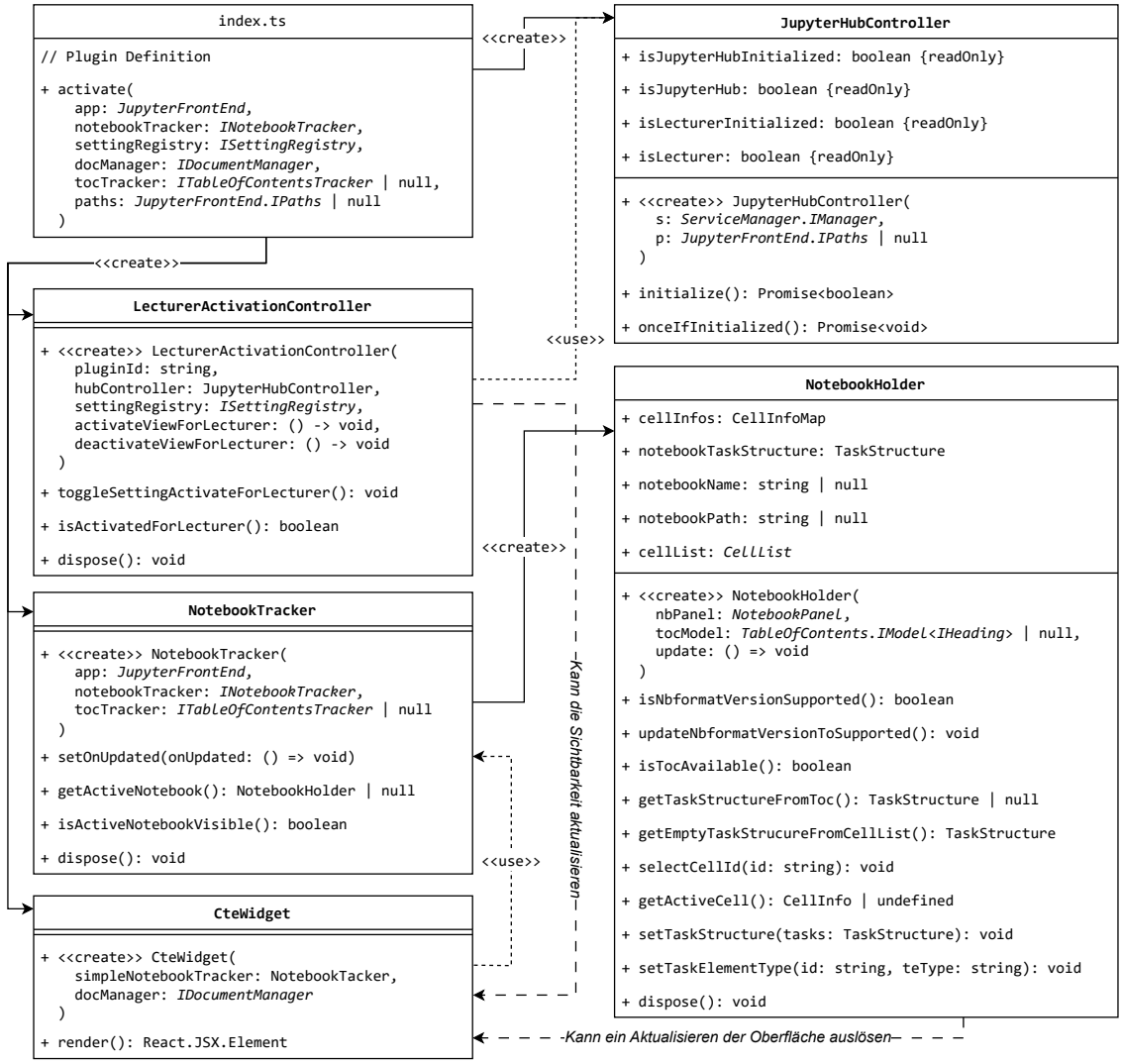



**Abbildung 4.8:** Übersicht über die im Plugin verwendeten Klassen und ihre Zusammenhänge. Es existiert maximal eine Instanz jeder Klasse. Die Typ-Definitionen, die in *Kursivschrift* dargestellt sind, stammen aus den `@jupyterlab`-Paketen. Eine Übersicht über die *React*-Komponenten wird in Abbildung 4.9 gegeben.

*Lecturer-Activation-Controller*

Diese Informationen werden vom `LecturerActivationController` genutzt, in welchem die Logik zur Aktivierung der Lehrendenansicht gekapselt ist. Wenn also ein *JupyterHub* verfügbar ist, aktiviert oder deaktiviert das Plugin die Lehrendenansicht basierend auf der Rolle der Nutzenden. Wenn dies nicht der Fall ist, wird die Aktivierung oder Deaktivierung anhand der dann sichtbaren Option in den Einstellungen des Plugins vorgenommen.

Die beiden Klassen `NotebookTracker` und `NotebookHolder` werden für die Verwaltung des aktiven *Notebook*-Dokumentes benötigt. Events werden in *JupyterLab* durch `Signals` realisiert, welche mit einem `Slot` abonniert werden können [44]. Ausgelöste Events sind zum Beispiel das Wechseln des geöffneten *Notebook*-Dokumentes oder der aktiven Zelle innerhalb eines Dokumentes.

*NotebookTracker*

Die Klasse `NotebookTracker` reagiert auf Wechsel des aktuell geöffneten *Notebook*-Dokumentes und gibt der Oberfläche des Plugins Bescheid, dass sich die Daten aktualisiert haben. Bei einem Wechsel oder Öffnen eines *Notebook*-Dokumentes wird außerdem ein Objekt der Klasse `NotebookHolder` für das aktive *Notebook*-Dokument erzeugt.

*NotebookHolder*

Die Klasse `NotebookHolder` dient als Wrapper für ein geöffnetes *Notebook*-Dokument. Dabei werden Slots für Änderungen an der Liste der Zellen im Dokument, den Zellinhalten und an den Metadaten registriert, sodass diese zu einem Aktualisieren der angezeigten Datenstruktur führen. Sobald ein anderes *Notebook*-Dokument geöffnet wird, kann der alte `NotebookHolder`-Wrapper durch die Funktion `dispose` aufgeräumt werden, wodurch alle registrierten `Slots` wieder entfernt werden.

Das Aktualisieren der *Notebook*-Dokumenten Version auf 4.5 (A4) kann durch den Aufruf von `updateNbformatVersionToSupported()` ebenfalls erledigt werden. Ebenso wird hier auch die Erzeugung einer initialen Aufgabenstruktur zur Verfügung gestellt. Diese kann dabei mit Hilfe des Inhaltsverzeichnisses erzeugt werden, sofern der Extension Point `@jupyterlab/toc:ITableOfContentsTracker` in *JupyterLab* zur Verfügung steht [29].

*DataStructureCellListComparator*

Die zwei Klassen `DataStructureCellListComparator` und `DataStructureExporter` werden direkt von den *React*-Komponenten verwendet und sind deswegen in der Übersicht in Abbildung 4.8 nicht vorhanden. Die Klasse `DataStructureCellListComparator` kann die Aufgabenstruktur reparieren (A5). Dafür wird ein Objekt der Klasse mit der Zellliste und der dazugehörigen aktuellen Aufgabenstruktur aus einem *Notebook*-Dokument initialisiert. Dieses Objekt stellt dann zwei öffentliche Methoden zur Verfügung. Die Methode `check()` gibt an, ob ein Fehler zwischen den beiden Datensätzen vorliegt. Mit `repair()` wird eine neue, reparierte Aufgabenstruktur zurückgegeben, welche mit den Zellen aus einem *Notebook*-Dokument übereinstimmt.

*DataStructureExporter*

Die Klasse `DataStructureExporter` gibt für ein vollständig klassifiziertes *Notebook*-Dokument die dazugehörige Datenstruktur als XML-String zurück, welche dann abgespeichert werden kann.

### React-Komponenten

Die Oberfläche des Plugins besteht aus *React*-Komponenten, die wie in Abbildung 4.9 dargestellt verschachtelt sind. Das `CteWidget` initialisiert zwei *React*-Komponenten in die Seitenleiste. Die beiden Komponenten stellen die Aufgabenstruktur und die Metadaten der aktiven Zelle dar.

*Gruppe: Aufgabenstruktur*

Die obere Gruppe `NbTaskStructureGroup` ist für die Klassifizierung und Darstellung der Aufgabenteile in Aufgaben und Teilaufgaben zuständig. Sobald eine Datenstruktur vorhanden ist, wird der `TaskStructureWrapper` angezeigt, welcher den Export-Button und die Darstellung der Aufgabenstruktur enthält. Diese Darstellung wird durch rekursives Verwenden der `TaskContainer`-Komponente erzeugt, welcher eine Aufgabe mit den jeweiligen Aufgabenelementen und Teilaufgaben darstellt.

Die Aufgabenüberschriften mit den Editieroptionen werden durch den `TaskHeader` und die Aufgabenelemente durch das `TaskElement` dargestellt. Die Komponente `AddNewSubTaskButton` erzeugt die Oberfläche zum Hinzufügen von neuen Teilaufgaben, welche angezeigt wird, sobald man mit der Maus darüberfährt.

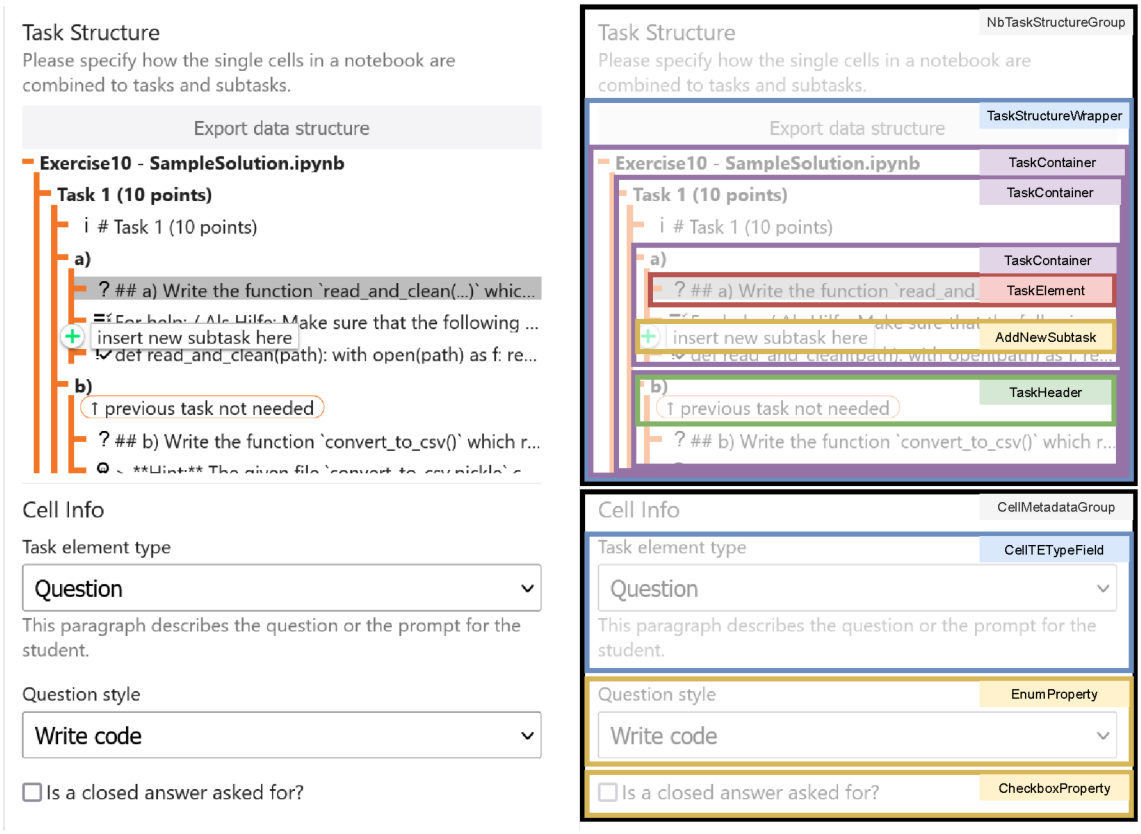



**Abbildung 4.9:** Auf der linken Seite ist die Oberfläche und auf der rechten Seite die Aufteilung in die *React*-Komponenten dargestellt. Zur besseren Übersichtlichkeit wurden nicht alle Vorkommen der Komponenten markiert.

*Gruppe: Zell-Metadaten*

Die untere Gruppe `CellMetadataGroup` dient der Klassifizierung der Aufgabenelemente. Die Komponente `CellTETypeField` stellt ein Auswahlformularfeld dar, welches je nach ausgewähltem Typ zusätzlich die Beschreibung des Typs anzeigt.
Weitere Attribute werden je nach ausgewähltem Typ des Aufgabenelementes unterhalb der Komponente `CellTETypeField` durch je eine der Komponenten `CheckboxPropertyField`, `EnumPropertyField` und `StringProtertyField` dargestellt. Das passende `PropertyField` wird je nach unterstütztem Datentyp des Attributes ausgewählt.

### Definition der Typen der Aufgabenelemente

*Flexibilität des Plugins*

Die möglichen Typen der Aufgabenelemente und ihre jeweiligen Attribute sind in eine eigene Datei ausgelagert worden, um eine einfachere Wartbarkeit zu gewährleisten. Die Datei `taskElementDefinitions.ts` im Ordner `res/` enthält eine Repräsentation der erarbeiteten Datenstruktur. Diese wird vom Plugin verwendet, um die Komponenten in der unteren Gruppe anzuzeigen, die zur Bearbeitung der Eigenschaften eines Aufgabenelementes verwendet werden. Dadurch lässt sich das Plugin einfach an Verbesserungen der Datenstruktur anpassen.

# Kapitel 5 Evaluation

Sowohl die erarbeitete Datenstruktur als auch das Plugin werden in den folgenden Abschnitten evaluiert. Für die Evaluation wurde eine Umfrage mit verschiedenen Personen aus dem Lehrbetrieb durchgeführt. In Abschnitt 5.1 wird die dafür erstellte Umfrage beschrieben und ihr Zweck erläutert. Die Ergebnisse der Umfrage werden in Abschnitt 5.2 ausgewertet und diskutiert. Zum Abschluss wird in Abschnitt 5.3 die überarbeitete Version der Datenstruktur präsentiert.

## 5.1 Aufbau und Zweck der Umfrage

*Ziel*

Die Umfrage hat zwei Ziele. Einerseits evaluiert sie die Nutzungsfreundlichkeit der Oberfläche des Plugins und erfasst Verbesserungspotentiale. Andererseits überprüft sie die Verständlichkeit der Datenstruktur und der Aufgabenelementtypen.
Die Umfrage wurde online durchgeführt und dabei über *Zoom*[1] begleitet. Sie besteht aus zwei Teilen und wurde in einer deutschen und englischen Version zur Verfügung gestellt. Der vollständige Fragebogen befindet sich in Anhang C und D für die deutsche und englische Version.

### 5.1.1 Plugin testen

Im ersten Teil wird in das Thema eingeleitet und die verschiedenen Aufgabenelementtypen vorgestellt. Anschließend soll das Plugin dort von den Teilnehmenden getestet werden. Dafür wurde eine *JupyterLab*-Instanz mit dem installierten Plugin und einem Übungsblatt des Kurses P4DS[2] über eine Bildschirmfreigabe mit Maus- und Tastatur-Kontrolle über *Zoom* zur Verfügung gestellt.
Zum Testen des Plugins sollten die Teilnehmenden die folgenden sechs Punkte durcharbeiten und Anmerkungen, die ihnen direkt in den Sinn kommen, notieren:

1. Generieren Sie die Aufgabenstruktur aus dem Inhaltsverzeichnis.
2. Wie Sie sehen können, wurde der Titel des Übungsblattes fälschlicherweise als Aufgabe interpretiert. „Introduction to Programming for Data-driven Sciences“ sollte jedoch keine eigenständige Aufgabe sein. Diese Aufgabe sollte daher entfernt werden.
3. Entfernen Sie ebenfalls die Angabe der erreichbaren Punkte aus dem Titel der Aufgabe in der Struktur.

[1] https://zoom.us
[2] Einführung in die Programmierung für datenbasierte Wissenschaften

4. Für die Aufgabe 1 a) wird die Datei „read_and_clean.txt“ benötigt und für die Aufgabe 1 b) die Datei „convert_to_csv.pickle“. Bitte verknüpfen Sie die vorhandenen Dateien mit der jeweiligen Aufgabe.

5. Geben Sie bei Aufgabe 2 c) an, dass sie auf dem Ergebnis von 2 a) und b) aufbaut.

6. Klassifizieren Sie nun die Aufgabenelemente bzw. Zellen aus Aufgabe 2 b) und füllen Sie gegebenenfalls die weiteren Attribute zur jeweiligen Klassifizierung aus.

Mit dieser Aufgabenliste werden die Teilnehmenden durch die wichtigsten Funktionen des Plugins geleitet. Außerdem wird dadurch sichergestellt, dass die Befragten für die spätere Beantwortung der Fragen den gleichen Wissensstand in Bezug auf die Benutzung der Funktionen des Plugins haben.

### 5.1.2 Fragen beantworten

Im zweiten Teil wurden dann verschiedene Daten zu der Nutzbarkeit des Plugins, der Verständlichkeit der Datenstruktur und zu den Teilnehmenden selbst erhoben.

*System Usability Scale*

Direkt nach dem Testen des Plugins wird der *System Usability Scale (SUS)* durch die entsprechenden Fragen erhoben, welcher ein etablierter Standard zum Bewerten der Nützlichkeit von Systemen ist [4]. Für die deutsche Übersetzung wurden die Fragen von Gao et al. verwendet [9].
Durch den *SUS-Score* lassen sich verschiedene Varianten eines Tools in ihrer Nutzungsfreundlichkeit vergleichen. Brooke gibt für den *SUS-Score* jedoch keine absolute Wertung [4] an. Um diese Lücke zu schließen, haben Bangor et al. für verschiedene Tools den *SUS-Score* in Umfragen ermittelt und zusätzlich die Befragten die allgemeine Nutzungsfreundlichkeit des Tools auf einer siebenstufigen Likert-Skala von Adjektiven einordnen lassen [2]. Dadurch lässt sich der *SUS-Score* in der Auswertung einordnen.
Anschließend werden konkrete Aussagen zu den einzelnen Funktionen des Plugins und deren Verwendung präsentiert. Zu jeder Aussage konnte eine Antwort auf einer fünfstufigen Skala von „Stimme überhaupt nicht zu“ bis „Stimme voll und ganz zu“ abgegeben werden. Zusätzlich hatten die Teilnehmenden hier die Möglichkeit, in einem Freitextfeld Funktionen zu benennen, die für sie nicht verständlich waren oder eine weitere Erläuterung erforderten.

*Evaluation zur Datenstruktur*

Anschließend wurden wieder Aussagen mit der gleichen fünfstufigen Skala präsentiert. Die Aussagen beziehen sich nun jedoch auf die Aufgabenelementtypen der Datenstruktur. Das Ziel dieser Aussagen ist es, die erarbeitete Aufteilung zu überprüfen und zum Beispiel zu ermitteln, ob möglicherweise eine feinere oder gröbere Aufteilung sinnvoll ist. Auch wird hier erfragt, ob sich die Aufgabenelementtypen für die Teilnehmenden in ihrer Definition überschneiden oder ob ihnen Typen fehlen. Zusätzlich zu den Aussagen, die über die fünfstufige Skala beantwortet wurden, gab es ein weiteres Freitextfeld, in welches Anmerkungen zu der Aufteilung gegeben werden konnten. Dort wurden zum Beispiel die Typen benannt, bei welchen eine Überschneidung in der Definition wahrgenommen wurde.

*Weitere Verwendungsmöglichkeiten*

Nachdem sich die Teilnehmenden nun mit dem Plugin und der Datenstruktur auseinandergesetzt haben, wurden sie gefragt, ob sie sich vorstellen können, dass die Klassifizierung der Aufgabenelemente auch in anderen Bereichen hilfreich sein kann. Bei der Durchführung der Umfrage musste diese Frage jedoch häufiger erläutert werden, da es oft unklar war, was mit „anderen Bereichen“ gemeint ist. Zur Erklärung wurde gesagt, dass aktuell die Verwendung im Bereich der Feedback-Generierung angedacht ist.

*Demografische Daten*

Zum Abschluss der Umfrage wurden die Teilnehmenden gebeten, noch ein paar Angaben zu sich zu machen, damit die erhobenen Daten eingeordnet werden können. Dabei sollten sie ihr ungefähres Alter und ihre Rolle (Dozent*in, Tutor*in, Student*in, ...) im Lehrbetrieb angeben. Zusätzlich wurde ihr allgemeiner Bezug zu Übungsblättern erfasst, das heißt, ob die Teilnehmenden Übungsblätter bearbeitet, erstellt und/oder schon korrigiert haben oder die Ergebnisse der korrigierten Übungsblätter sogar schon für weitere Zwecke benutzt haben. Die letzte Frage erfasst, wie oft die Teilnehmenden bereits mit *JupyterLab* gearbeitet haben. Dadurch soll ermittelt werden, ob Unklarheiten in der Oberfläche des Plugins möglicherweise auf mangelnde Erfahrung mit *JupyterLab* zurückzuführen sind.

## 5.2 Auswertung der Ergebnisse

*Angaben zu den Teilnehmenden*

Es nahmen acht Personen im Alter von 20 bis 29 Jahren teil. Die Umfrage inklusive des Testens des Plugins dauerte im Schnitt jeweils ca. 30 Minuten. Wie Abbildung 5.1 zeigt, bearbeiten fast alle regelmäßig Übungsblätter und jeweils drei Personen haben auch schon Übungsblätter erstellt oder korrigiert. Zwei der erstellenden Personen nutzen die Ergebnisse der korrigierten Übungsblätter. Zusätzlich wurden gefragt, welcher Gruppe sich die Teilnehmenden zugehörig fühlen. Sechs Teilnehmende gaben Student*in an, eine Person Dozent*in, zwei Personen Tutor*in und einmal wurde „Übungsleiter“ (P8) angegeben. Auch hier war eine Mehrfach-Auswahl bei der Antwort möglich.

In Abbildung 5.2 ist zu sehen, dass die meisten Teilnehmenden die *JupyterLab*-Oberfläche schon mal verwendet haben. Fünf von acht Befragten gaben sogar an, die Oberfläche gelegentlich bis wöchentlich zu nutzen.

### 5.2.1 System Usability Scale

Die Ergebnisse der Berechnung des *System Usability Scale (SUS) Score*s sind in Abbildung 5.3 dargestellt. Sowohl der Durchschnitt als auch der Median liegen bei ca. 79 Punkten, jedoch sagt der Wert allein noch nichts über die Nutzungsfreundlichkeit aus. Mit Hilfe der Studienergebnisse von Bangor et al. lässt sich der *SUS-Score* von 79 Punkten im Bereich *Good* bis *Excellent* einordnen [2]. Der durchschnittliche *SUS-Score* ist 71,4 für den Bereich *Good* und 85,5 für den Bereich *Excellent* [2]. Sogar der Bereich zwischen den beiden erhobenen Quartilen (71,3 und 86,3) liegt fast vollständig innerhalb dieser beiden Kategorien.

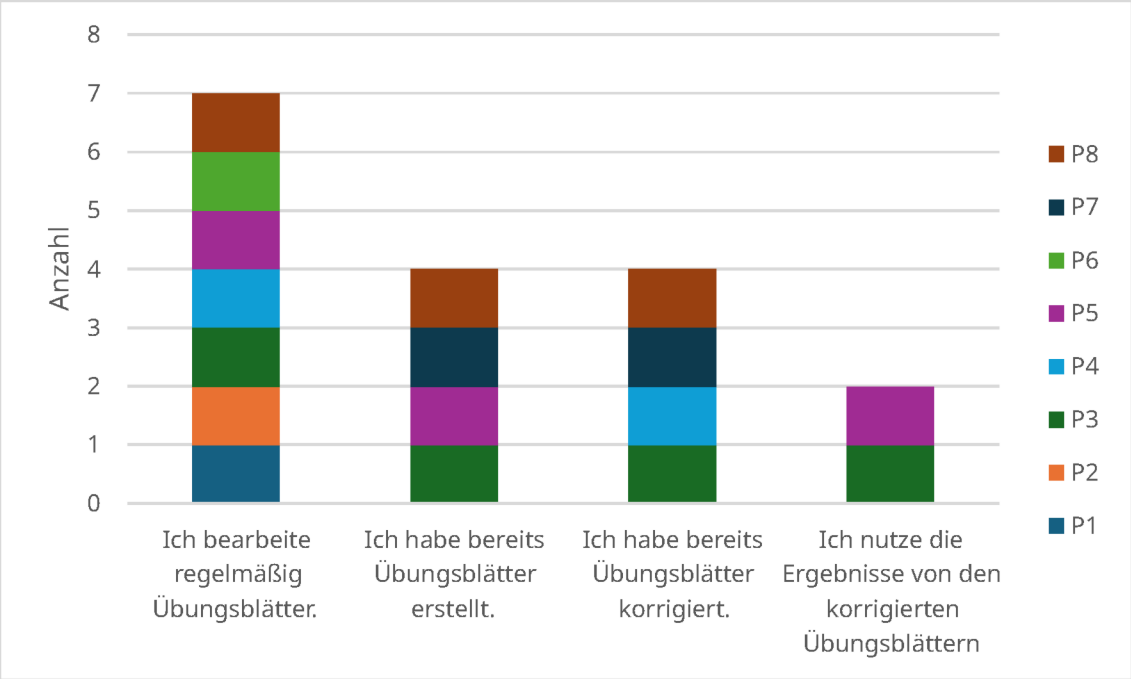



**Abbildung 5.1:** Antworten der Teilnehmenden, um ihren Bezug zu Übungsblättern zu erfahren. Mehrfach-Auswahl möglich. Jede Farbe steht für eine Person.

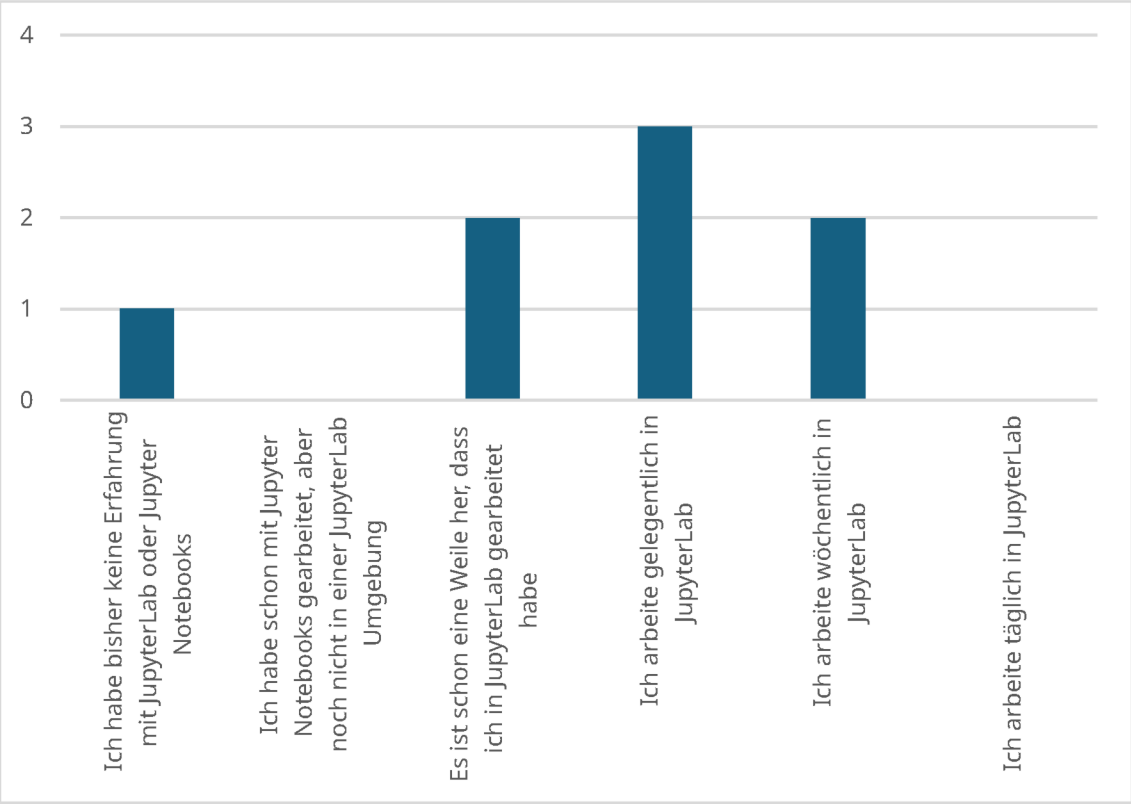



**Abbildung 5.2:** Antworten auf die Frage „Wie oft arbeiten Sie mit Jupyter-Notebooks?“ Keine Mehrfach-Auswahl möglich.

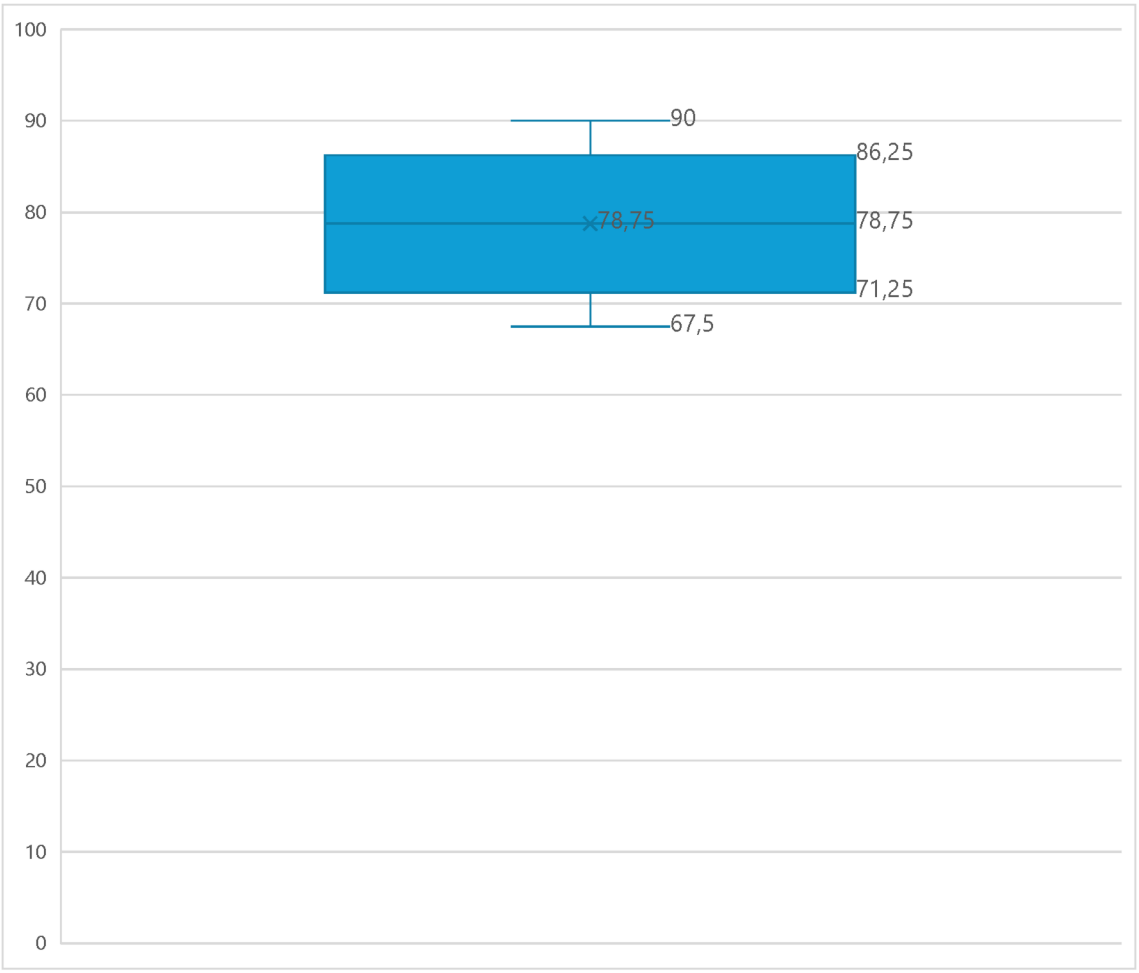



**Abbildung 5.3:** Auswertung der berechneten *SUS-Score*s.

Daraus lässt sich schließen, dass das Plugin generell nutzungsfreundlich gestaltet ist (Anforderung A2). Aus den Antworten in den Freitextfeldern lassen sich weitere Verbesserungen ableiten, welche in Unterabschnitt 5.2.2 diskutiert werden. Diese scheinen jedoch nicht besonders die Nutzungsfreundlichkeit negativ beeinflusst zu haben.
Wenn man die Angaben der Teilnehmenden in Gruppen nach ihrer Nutzungshäufigkeit von *JupyterLab* aufgeteilt betrachtet, weicht dort nur die Gruppe „Ich habe bisher keine Erfahrung mit JupyterLab oder Jupyter Notebooks“ stärker ab. Für diese Gruppe wird ein Mittelwert von 75 Punkten erreicht, welcher auch ein gutes Ergebnis für die Nutzungsfreundlichkeit darstellt.

### 5.2.2 Evaluation des Plugins

In den folgenden Abschnitten erfolgt für die Aussagen zu den einzelnen Plugin-Funktionen eine Auswertung, inwieweit die Teilnehmenden diesen zugestimmt haben.

*Aufgabenstruktur erfassen und anpassen*

Die automatische Erfassung der Aufgabenstruktur wurde von allen Teilnehmenden als hilfreich angesehen. Wie in Abbildung 5.4a zu erkennen ist, wurde dort mindestens „Stimme zu“ oder besser angegeben. Auch zur Aussage „Das Anpassen der Aufgabenstruktur ist intuitiv gestaltet“ stimmen die meisten Teilnehmenden zu, wobei sich

zwei Teilnehmende für die neutrale Option „Stimme weder zu noch lehne ich ab“ entschieden haben, wie in Abbildung 5.4b zu sehen ist. Generell ist also die Erstellung und die Veränderung der Aufgabenstruktur durch das Plugin für die Befragten gut durchführbar gewesen.

Jedoch wurde in den Freitextfeldern angegeben, dass es gerade beim Schritt zum Löschen der falsch erfassten Aufgabe in der Struktur zu Schwierigkeiten kam. Hierbei sollte die automatisch erfasste Struktur angepasst werden. Die meisten Teilnehmenden haben erwartet, dass durch das Löschen der Aufgabe auch die dazugehörigen Zellen im *Jupyter-Notebook*-Dokument gelöscht werden. Dies geht aus den folgenden Angaben hervor:

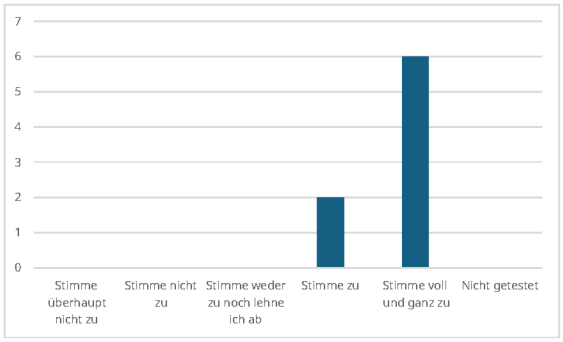
- „Ich habe zuerst vermutet, dass die Hierarchie rekursiv gelöscht werden würde.“ (P1)
- „Etwas unintuitiv, da das löschen für mich ein bearbeiten im notebook suggeriert [sic]“ (P8)
- „Das Löschen von higher-level Elementen, wie in dem Fall der Aufgabe, scheint nicht das Löschen von Subelementen zur Folge zu haben. Das erscheint mir unintuitiv und könnte klarer kommuniziert werden.“ (P4)

Durch das Plugin ist keine Bearbeitung der *Notebook*-Dokumente vorgesehen, sondern nur die Erfassung der Datenstruktur zur Klassifizierung. Deshalb ist der Vorschlag des letzten Kommentars sinnvoll. Nach einmaligem Verwenden ist die Funktion zwar klar, aber durch eine vorherige Kommunikation können Missverständnisse besser vermieden werden. Im Allgemeinen wird die Funktion zum Löschen nicht einfach ausprobiert. Stattdessen möchte man vorher wissen, was dadurch tatsächlich gelöscht wird.

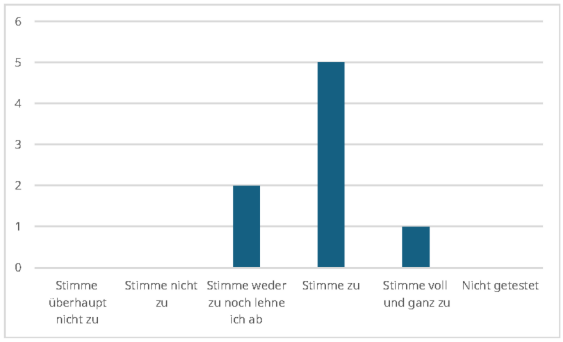
*Abhängigkeiten angeben*

Während des Testens sollten ebenfalls Abhängigkeiten zwischen zwei Teilaufgaben in der Aufgabenstruktur angegeben werden. Hierbei wurde angegeben, dass recht gut verstanden wurde, wie dies in der Oberfläche gemacht werden kann. Die Auswertung in Abbildung 5.4c zeigt, dass nur eine einzige Person die neutrale Option gewählt hat, während alle anderen der Aussage zugestimmt haben. Dieses eher gute Ergebnis bezieht sich jedoch nur auf die Oberfläche. Das Verständnis dessen, was eine Abhängigkeit bedeutet, ist jedoch nicht direkt gegeben gewesen, wie die Auswertung in Abbildung 5.6c zeigt.

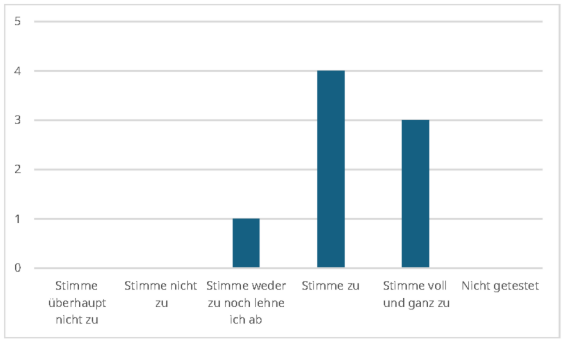
Bis auf eine Ausnahme war es für die Befragten auch gut möglich, benötigte Dateien mit der jeweiligen Aufgabe zu verknüpfen (siehe Abbildung 5.5a). Im vorliegenden Fall wurden die Symbole zur Bearbeitung der Aufgabe nicht gefunden. Während des Plugin-Tests wurde die befragte Person darauf hingewiesen, dass ein Verlinkungssymbol erscheint, wenn man die Maus über eine Aufgabe hält. Zuvor wurde versucht, die Datei aus dem Dateimanager auf der linken Seite einfach auf eine Aufgabe oder das Aufgabenelement mit der Fragestellung zu ziehen.




***(a)*** Die automatische Erfassung der Struktur ist hilfreich.




***(b)*** Das Anpassen der Aufgabenstruktur ist intuitiv gestaltet.



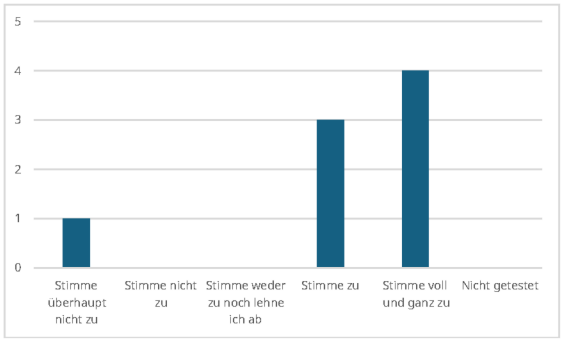

***(c)*** Ich habe verstanden, wie ich die Abhängigkeit einer Aufgabe von der vorherigen Aufgabe angeben kann.

**Abbildung 5.4:** Auswertung der Aussagen zum Plugin, die von den Teilnehmenden eine hohe Zustimmung erhalten haben.

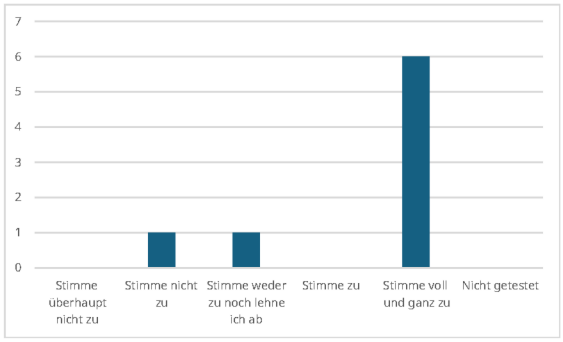
Die gleiche Person hat auch angegeben, dass sie der Aussage „Ich habe schnell herausgefunden, wie ich den Namen einer Aufgabe in der Struktur ändern kann“ nicht zustimmt. Da dies auch mit denselben Symbolen gemacht werden musste, hat es vermutlich ebenfalls den Grund, dass die Symbole nicht gesehen wurden. Eine andere Person hat hier jedoch ebenfalls nicht zugestimmt, sondern mit der neutralen Option geantwortet, wie in Abbildung 5.5b zu sehen ist. Da alle anderen Befragten „voll und ganz“ zugestimmt haben, kann davon ausgegangen werden, dass die Möglichkeit zum Bearbeiten des Namens leicht zu finden ist.

*Aufgabenelementtyp zuweisen*

Es gab aber auch ein paar Aussagen, denen nicht so häufig zugestimmt wurde, und die dann auf mögliche Probleme der Funktionen im Plugin hinweisen. Wie in Abbildung 5.6a zu sehen, ist das Zuordnen der Aufgabenelementtypen zu einer Zelle für mindestens zwei Personen nicht intuitiv. Dies kann zwei Gründe haben. Erstens sind die zur Auswahl stehenden Kategorien für die Befragten nicht eindeutig oder passend gewesen, d.h. es gab Probleme mit dem Verständnis der Datenstruktur. Oder es liegt zweitens daran, dass die Oberfläche so strukturiert war, dass die Funktion zum Zuordnen nicht gefunden wurde.
Eine Auswertung der Freitext-Antworten ergab, dass letzteres der Fall ist. Es wurde von verschiedenen Teilnehmenden berichtet, dass die Auswahl im unteren Bereich nicht intuitiv ist, wenn man vorher die Zelle oben in der Aufgabenstruktur auswählt. Als Verbesserung gibt es den Vorschlag, die „Cell Info“ (P2) mehr hervorzuheben. Dies




***(a)*** Es war für mich einfach, die passenden Dateien der jeweiligen Aufgabe anzuhängen.




***(b)*** Ich habe schnell herausgefunden, wie ich den Namen einer Aufgabe in der Struktur ändern kann.

**Abbildung 5.5:** Auswertungen der Aussagen zum Plugin, bei denen die Teilnehmenden eher zugestimmt haben, obwohl es auch einzelne Ausreißer gab.

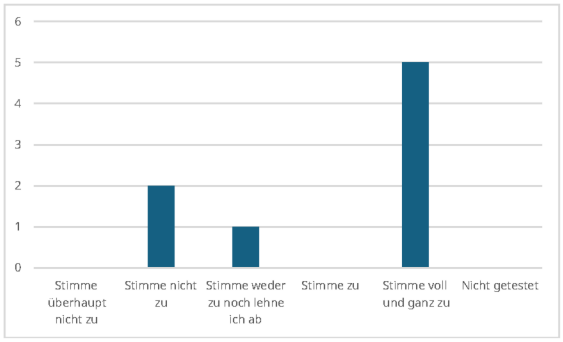
kann zum Beispiel durch den Anfang des Zellinhalts geschehen, da sich dieser beim Ändern der markierten Zelle ebenfalls ändert und somit zu einer erhöhten Aufmerksamkeit bei den Nutzenden führen kann. Ebenso wäre dadurch direkt erkennbar, auf welche Zelle sich die Änderungen beziehen.
Alternativ wurde auch erwartet, dass eine Bearbeitung der Zellinformationen genauso funktioniert wie eine Bearbeitung einer Aufgabe. Zwei Befragte gaben an, dass sie beim Hovern ein Stiftsymbol oder andere Elemente zum Bearbeiten erwartet hätten. Ein Rechtsklick wurde ebenfalls versucht, führte jedoch nicht zum gewünschten Ergebnis.

*Zugehörigkeit von Zellen zu Aufgaben*

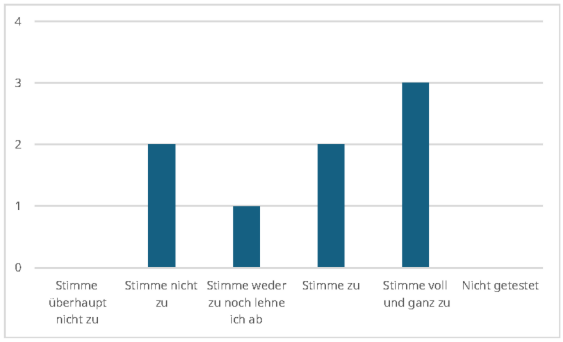
Obwohl die Anpassbarkeit der Aufgabenstruktur als intuitiv bewertet wurde, scheint es Schwierigkeiten gegeben zu haben, auf Anhieb zu erkennen, welche Aufgabenelemente bzw. Zellen zu einer Aufgabe gehören (siehe Abbildung 5.6b). Durch Übersetzung der Kategorien in die Werte eins bis fünf lässt sich ein Mittelwert berechnen, welcher dann bei 3,75 liegt. Dies liegt zwischen der neutralen Option mit Wert 3 und der Option „Stimme zu“ mit Wert 4. Es gibt somit noch eine leichte Tendenz zur Zustimmung, jedoch ist sie nicht so eindeutig wie bei den anderen Aussagen.
Zu diesem breit gestreuten und eher schlechteren Ergebnis wurden leider auch keine weiteren Details in den Freitextfeldern hinterlassen, die Hinweise geben, wie eine Darstellung verbessert werden könnte. Der einzige Kommentar dazu war „[Die Aufgabe] war einfach zu löschen, die Struktur der Aufgabe war nur nicht direkt zu erkennen“ (P6). Deshalb können hier nur Vermutungen angestellt werden, was zu dieser Bewertung geführt hat. Die Baumstruktur-Darstellung wird oft in Dateiexplorern genutzt. Allerdings kann sie durch die zusätzlichen vertikalen Linien zu stark abweichen und somit schwerer verständlich sein. Möglicherweise haben die Teilnehmenden auch nicht erkannt, dass die angezeigten Aufgabenelemente in der Struktur mit den vorhandenen Zellen im *Notebook*-Dokument zusammengehören.

*Abhängigkeit von Aufgaben verstanden*

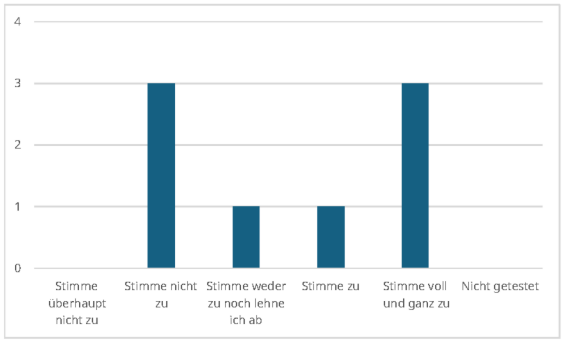
Obwohl die meisten verstanden haben, wie sie die Abhängigkeit von Aufgaben in der Oberfläche angeben (siehe Abbildung 5.4c), scheint es Schwierigkeiten bei dem Verständnis gegeben zu haben. Mit einem Mittelwert von 3,5 ist dies die Aussage mit der geringsten Zustimmung (siehe Abbildung 5.6c). Dabei wird der Option „Stimme




***(a)*** Das Zuweisen eines Typs zu einer Zelle war für mich intuitiv.




***(b)*** Es war direkt ersichtlich, welche Aufgabenelemente/Zellen zu einer Aufgabe in der Struktur gehören.




***(c)*** Ich habe verstanden, was eine Abhängigkeit zwischen einer Aufgabe und der vorhergehenden Aufgabe bedeutet.

**Abbildung 5.6:** Auswertungen der Aussagen zum Plugin, bei denen die Angaben der Teilnehmenden auseinander gehen.

überhaupt nicht zu“ der Wert 1 zugewiesen. Die Option „Stimme weder zu noch lehne ich ab“ entspricht dann dem Wert 3 und der Mittelwert ist somit nur knapp drüber. Dennoch zeigt das Ergebnis, dass eine klarere Kommunikation über die Definition der Abhängigkeit von (Teil-)Aufgaben notwendig ist.

*Abhängigkeiten nicht aufbauend*

Dieses Problem wurde auch in den Freitextfeldern angesprochen. Mehrere befragte Personen wiesen darauf hin, dass in der aktuellen Umsetzung der Datenstruktur nicht klar sei, worauf sich eine Abhängigkeit bezieht. „Impliziert die Einstellung bei c), dass es nur von b) abhängt oder von allen davor (also auch a)?“ (P4)

*Tooltips*

Über die Angaben in den Freitextfeldern gaben die Teilnehmenden noch weiteres Feedback zum Plugin. Die Teilnehmenden äußerten am häufigsten den Wunsch nach Tooltips an den Buttons. Dies würde auch bei dem Löschen-Button an einer Aufgabe in der Struktur helfen, da hier direkt kommuniziert werden kann, dass die Zellen im *Notebook*-Dokument nicht verändert werden. Aber auch die anderen Symbole zum Bearbeiten können von Tooltips profitieren. Eine teilnehmende Person hat es gut auf den Punkt gebracht: „Die Symbole sind zwar irgendwo erklärend, gerade wenn man das System kennt auch mit Sicherheit ausreichend, aber wenn man es noch nicht kennt sind kleine Texte, die sagen, was nach dem klicken [sic] passiert, nicht verkehrt.“ (P3)

*Tutorial*

Zusätzlich zu den Hinweisen an den Buttons wurde von zwei Befragten noch ein weiterer Vorschlag eingebracht, nämlich bei der ersten Verwendung ein Tutorial anzubieten, welches die Funktionen erläutert. Die Beschreibung der Buttons kann durch diese Variante noch detaillierter als durch Tooltips erfolgen. Die Oberfläche wurde als schnell erlernbar wahrgenommen, so dass es „beim zweiten verwenden [sic] nämlich überhaupt kein Problem mehr“ (P2) darstellt.

*Übersicht der Typen mit Erklärungen*

Ein weiterer Vorteil eines Tutorials ist die Möglichkeit, die verschiedenen Aufgabenelementtypen zu erläutern. Eine Übersicht der Typen mit einer kurzen Beschreibung wurde den Teilnehmenden in der Umfrage zur Verfügung gestellt. Sobald ein Typ in dem Plugin ausgewählt wurde, wurde dieser dort auch erläutert. Jedoch wurde angegeben, dass die Übersicht aus der Umfrage sehr geholfen hat. Eine Möglichkeit, die Beschreibungen in das Plugin zu integrieren, wäre die Anpassung der Auswahlliste. In der Liste sollten nicht nur die Titel der Typen angezeigt werden, sondern auch die dazugehörigen kurzen Beschreibungen.

*Text-Verbesserungen*

In den Freitextantworten wurden auch weitere kleinere textuellen Verbesserungsvorschläge für das Plugin gemacht. Es wurde vorgeschlagen, den Button zum Erstellen der initialen Aufgabenstruktur aus dem Inhaltsverzeichnis ausführlicher zu beschriften, zum Beispiel mit „Generate Task Tree from Structure“ (P5) statt „Create from Table of Contents“. Außerdem ist die Abkürzung „desc.“, welche im Plugin für „description“ genutzt wird, nicht direkt ersichtlich und sollte an der verwendeten Stelle ausgeschrieben werden.

*Automatisches Anpassen der Titel*

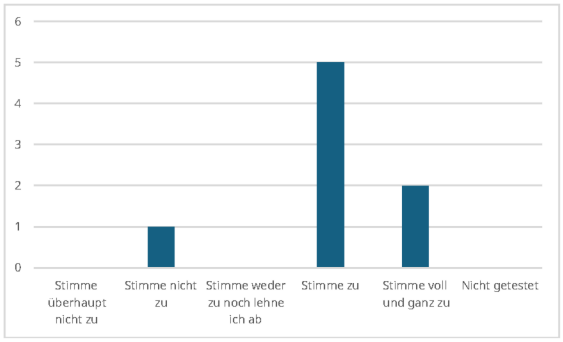
Eine befragte Person schlug noch vor, dass es praktisch wäre, wenn bei der automatischen Erkennung der Aufgabenstruktur die Aufgabentitel direkt angepasst werden würden. Aktuell nutzt das Plugin die Überschrift als Aufgabentitel, sodass bei dem zur Verfügung gestellten Übungsblatt im Aufgabentitel auch die maximale Punktzahl in Klammern landete. Dies sollte anschließend manuell entfernt werden. Hier wurde die Überlegung angestellt, ob sich der letzte Schritt auch automatisieren ließe.

*Design*

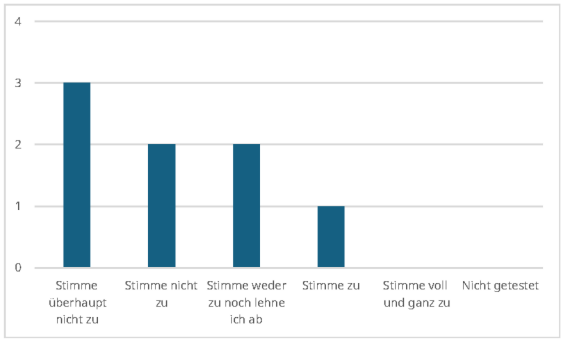
Des Weiteren wurden Vorschläge zur Optimierung des Designs formuliert. So wurde beispielsweise angeregt, die Symbole für die Typen der einzelnen Aufgabenelemente stärker hervorzuheben. Dies kann zum Beispiel durch eine farbliche Hervorhebung oder dickere Linien umgesetzt werden. Zusätzlich wurde von einer anderen Person noch etwas mehr Abstand zwischen den einzelnen Aufgaben für eine bessere Übersichtlichkeit gewünscht.

Außerdem wurde der Button zum Generieren der Datenstruktur aus dem Inhaltsverzeichnis von einer teilnehmenden Person nicht als aktiv wahrgenommen, da er einen grauen Farbton besitzt. Dies ist das Standard-Design von *JupyterLab-Buttons*, welches als Richtlinie dient. Jedoch gibt es dort auch in Dialogen farbige Buttons, welche meist die gewünschte Aktion auslösen, während die Standard-Buttons den Vorgang meist abbrechen. Dieser farbige Button kann also auch an der Stelle zum Generieren der Aufgabenstruktur im Plugin verwendet werden, da es die gewünschte Aktion ist.

Ebenso wurde angegeben, dass der Button zum Angeben der Abhängigkeit von Aufgaben zu vorherigen Aufgaben auch nicht unbedingt als Button erkannt wurde und eine weitere Erklärung benötigt wird, auf welche vorherige (Teil-)Aufgaben er sich




***(a)*** Die Zuordnung der Typen zu den Aufgabenelementen war einfach.




***(b)*** Ich denke, dass einige Aufgaben nicht in die Typen eingeteilt werden können.

**Abbildung 5.7:** Einteilung und Zuordnung der Aufgabenelementtypen.

bezieht. Es muss jedoch generell ein neues Design für diese Funktion erstellt werden, da die Evaluation der Datenstruktur ergibt, dass die aktuelle Angabe der Abhängigkeit von Teilaufgaben unzureichend ist.

### 5.2.3 Evaluation der Datenstruktur

Es wurde darauf hingewiesen, dass es nicht möglich ist, anzugeben, ob eine Teilaufgabe von einer anderen abhängt, die nicht die direkt vorhergehende Teilaufgabe ist. Als Beispiel zur Erklärung: Derzeit kann nicht angegeben werden, dass Teilaufgabe c) nur von a) und nicht von b) abhängig ist. Ebenso ist die Angabe der Abhängigkeit einer Teilaufgabe von mehreren Teilaufgaben aktuell nur transitiv möglich. Dies wurde als Feedback zu der Oberfläche gegeben, jedoch bezieht es sich eher auf die zugrunde liegende Datenstruktur.

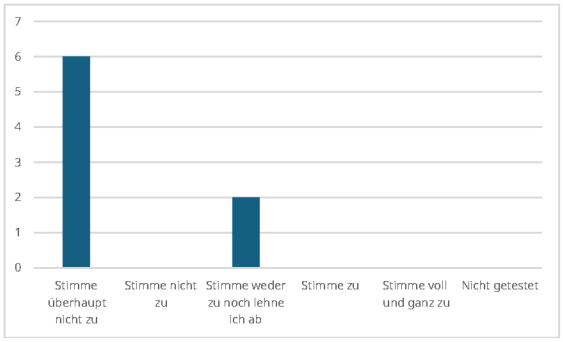
Allgemein bezogen sich die Fragen im Evaluationsbogen zur Datenstruktur jedoch eher auf die Klassifizierung der Aufgabenelemente und die erarbeiteten Typen von Aufgabenelementen. Die meisten Teilnehmenden stimmten dabei der Aussage zu, dass die Zuordnung zu den Aufgabenelementen einfach war (siehe Abbildung 5.7a). Nur eine Person stimmte nicht zu, jedoch gibt es dazu von ihr keine weiteren Kommentare.

*Missverständliche Formulierung der Aussage*

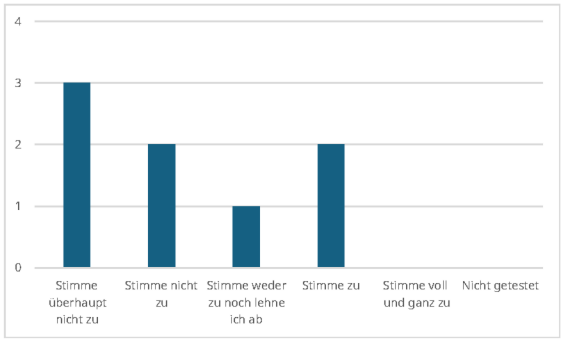
Von einer anderen Person gab es jedoch die Anmerkung, dass sie sich bei dem `hint`-Typ nicht sicher war und die Definition mehrmals durchlesen wollte. Da diese Person der Aussage jedoch generell zustimmte, wurde die Aussage möglicherweise so verstanden, dass das Zuweisen von Typen zu einem Aufgabenelement in der Oberfläche einfach ist. Tatsächlich sollte die Aussage jedoch erfragen, ob die Auswahl eines Typs für ein Aufgabenelement inhaltlich einfach war und nicht die Zuordnung über die Oberfläche.

*Nicht repräsentierbare Aufgaben*

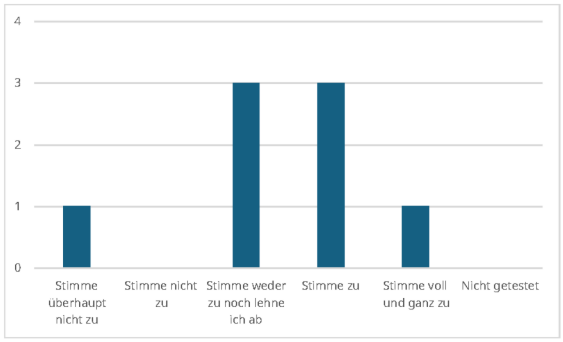
Die nächste Frage erfasst, ob alle Aufgabenelementtypen in der Datenstruktur vorhanden sind. Die Teilnehmenden wurden dabei gefragt, ob ihnen Aufgaben einfallen, welche sich nicht mit dem Schema klassifizieren lassen. In Abbildung 5.7b ist zu sehen, dass viele der Aussage widersprechen. Eine Person gab jedoch an, dass sie sich vorstellen könnte, dass einige Aufgaben nicht in die Typen eingeteilt werden können.




***(a)*** Ich habe mir oft eine feinere Unterteilung der Typen gewünscht.




***(b)*** Die Unterteilung in die vorgegebenen Typen war zu detailliert.




***(c)*** Es gab Überschneidungen zwischen den Typen.

**Abbildung 5.8:** Auswertungen zum Detailgrad der Aufteilung der Aufgabenelementtypen.

*Beantwortung zu aufwendig*

Zwei Befragte gaben auch an, dass sie der Aussage weder zustimmen noch sie ablehnen. Vermutlich war hier die Fragestellung zu offen gefasst. Gerade bei der Beantwortung von Fragebögen ist die Motivation meist nicht die größte, sodass ein umfangreiches Überlegen über verschiedene Aufgabenformate, welche nicht in die neue Struktur passen, möglicherweise zu aufwendig ist.

Dazu kommt, dass die verschiedenen Typen erst kurz vorher eingeführt wurden und sich die Teilnehmenden nicht umfassend damit beschäftigt hatten. Die Typen waren zu dem Zeitpunkt auch nicht mehr mit ihren Definitionen in der Umfrage sichtbar und keine der teilnehmenden Personen wechselte während der Bearbeitung zum Plugin zum Nachschlagen zurück. Die Ergebnisse zu dieser Aussage sind vor diesem Hintergrund entsprechend vorsichtig zu betrachten.

*Detailliert genug?*

Die Umfrage enthält jedoch nicht nur Fragen zu fehlenden Aufgabenelementtypen, sondern auch Fragen dazu, ob die vorhandenen Aufgabenelementtypen zu detailliert (siehe Abbildung 5.8b) oder zu unspezifisch (siehe Abbildung 5.8a) sind. Aus den Ergebnissen lässt sich entnehmen, dass eine feinere Unterteilung der Typen nicht gewünscht ist. Vermutlich kann es überfordernd sein, elf neue Typen auf einmal kennenzulernen, insbesondere wenn sie erst vor wenigen Minuten vorgestellt wurden.

*Zu detailliert?*

Deshalb wurde auch die gegenteilige Aussage erfasst, ob zu viele Kategorien zur Verfügung gestellt werden und ob eher eine weitere Reduzierung gewünscht wird. Die Zustimmungen sind, wie in Abbildung 5.8b zu sehen, eher gemischt. Insgesamt gaben jedoch fünf Personen an, dass sie der Aussage nicht oder überhaupt nicht zu stimmen.

Zwei Befragten war die Aufteilung zu detailliert und eine Person hat dazu auch eine Anmerkung hinterlassen. Darin wird betont, dass die Aufteilung beim ersten Nutzen unübersichtlich war, da sie sehr granular ist. Sie vermutete außerdem, dass man später eine „schnelle Eingewöhnung“ (P7) hat.

Mehr Typen sollten also vermieden werden und eine Reduzierung der Typen hilft den Lehrenden bei der ersten Benutzung. Langfristig kann die Liste auf Typen, die sich für die Generierung von Feedback als nützlich erweisen, reduziert werden. Andererseits kann eine Aufteilung in alle Typen notwendig sein, wenn unterschiedliche Typen für verschiedene Feedback-Generierungsmechanismen hilfreich sind.

*Überschneidungen*

Die Befragten gaben jedoch auch an, dass sich die vorgegebenen Typen aus ihrer Sicht überschneiden. In Abbildung 5.8c ist zu sehen, dass eine Person der Aussage widersprach, vier Personen stimmten jedoch zu. Diese Problematik wurde auch von allen Teilnehmenden in den Freitextfeldern spezifiziert, die der Aussage zugestimmt haben. Zusätzlich gab eine Person, welche sich für die Option „Stimme weder zu noch lehne ich ab“ entschieden hatte, noch eine weitere „potentielle Überschneidung“ (P4) an. Im Folgenden werden die angegebenen Überschneidungen präsentiert.

Die meisten Überschneidungen wurden für den Typ `hint` angegeben.

- „Im gegebenen Beispiel fiel mit [sic] die Klassifikation der Note schwer: Hint oder Constraint“ (P1)
- „Explanation und Hint fand ich teilweise recht ähnlich“ (P5)
- „Potentielle Überschneidung zwischen Hint und Selfttest. Im gegeben Beispiel war eine Zelle eine Kombination aus beidem. [sic]“ (P4)
- „hint/explanation kann beispiel [sic] enthalten“ (P8)

Aus den angegebenen Anmerkungen lässt sich ableiten, dass die Definition für den Typ `hint` überarbeitet werden sollte, da für die Teilnehmenden scheinbar nicht klar war, wie dieser von den anderen Typen abzugrenzen ist. Im Prinzip ist der `hint`-Typ dazu gedacht, dass weitere Hinweise zur Bearbeitung einer Aufgabe klassifiziert werden können. Wie der letzten Anmerkung zu entnehmen ist, wurde der `hint`-Typ jedoch eher als übergreifender Typ wahrgenommen.

*Verbesserung Typ `hint`*

Dieses Ergebnis kann zu einer Verbesserung der Datenstruktur und zu einer Vereinfachung des Klassifizierungsprozesses führen. Aus den vorherigen Antworten wurde herausgearbeitet, dass die vorhandene Unterteilung für einige Teilnehmende zu detailliert war. Dies lag vor allem daran, dass sie relativ viele neue Typen auf einmal kennen gelernt haben. Eine Möglichkeit zur Vereinfachung besteht darin, alle Typen, die als spezialisierte Varianten des Typs `hint` definiert werden können, aus der Typenklassifizierung zu entfernen und in weitere Eigenschaften des Typs `hint` umzuwandeln.

Dies kann zum Beispiel so aussehen, dass der Typ `hint` für die drei anderen Typen `constraint`, `explanation` und `example` jeweils ein boolesches Attribut besitzt. Dadurch steht effektiv noch die Möglichkeit zur Verfügung, die gleiche Klassifizierung vorzunehmen, wie sie bisher auch möglich war. Man wird jedoch weiterhin nicht direkt mit einer großen Auswahl an möglichen Typen überfordert, da einige Typen zu `hint` zusammengefasst werden. Zusätzlich ist es nun aber auch kein Problem, falls sich eine Aufgabe nicht in distinkte Aufgabenelemente mit je einem Typ aufteilen lässt, da es möglich wäre, eine Kombination der Typen auszuwählen. Dies würde die von Person P8 beschriebene Problematik lösen, dass ein Beispiel innerhalb von Aufgabenelementen des Typs `hint` oder `explanation` sein kann.

Jedoch ist die Anmerkung von Person P8 vermutlich eher auf ein Missverständnis der Definition des Typen `example` zurückzuführen. Dieser Typ ist ausschließlich für Beispiele gedacht, die alleinstehend einen Hinweis auf die grobe Strukturierung des Antwortcodes geben sollen. Beispiele, die zur Erklärung neuer Konzepte oder anderweitig als Beispiel genutzt werden, sind mit diesem Typ nicht gemeint.

Auf Grund der Anmerkungen zu den Überschneidungen bei manchen Typen und weil ein paar Befragten die Unterteilung in die Typen zu detailliert ist, wird hier eine Vereinfachung der Oberfläche vorgeschlagen. Dabei werden die vier genannten Typen zu dem Typ `hint` reduziert, jedoch kann dieser durch eine weitere Unterauswahl weiter spezifiziert werden. Der ursprüngliche Typ `hint` würde dann zum Beispiel durch die Auswahl „Hint“ und Unterauswahl „Other“ angegeben werden und der ursprüngliche Typ `explanation` durch eine Auswahl von „Hint“ und anschließend „Explanation“.

Theoretisch könnte im Hintergrund weiterhin die gleiche Datenstruktur erzeugt werden wie zuvor, während die Benutzeroberfläche vereinfacht wird. Allerdings kann eine Inkonsistenz der Darstellung und der Datenstruktur zu Verwirrung bei der weiteren Nutzung führen, da nicht bekannt ist, welche Darstellung in anderen Tools für die Datenstruktur genutzt wird. Daher wird eine Präsentation der Oberfläche bevorzugt, die der Datenstruktur möglichst nahekommt.

*Definition question*

Person P8 gab in der Evaluation auch an, dass der Typ `question` inhaltlich auch Elemente vom Typ `constraint` enthalten kann. Dies wurde in den überprüften *Jupyter-Notebook*-Dokumenten nicht wahrgenommen oder ließ sich immer in verschiedene Aufgabenelemente aufsplitten. Möglicherweise wurde hier der Typ `question` anders verstanden. Dies könnte zur Folge gehabt haben, dass zum Beispiel eine Beschreibung des Problems mit anschließender Fragestellung als ein Aufgabenelement vom Typ `question` klassifiziert wurde.

Jedoch ist hier die Beschreibung als eigenes Aufgabenelement vom Typ `constraint` zu verstehen und nur die wirkliche Frage oder Aufforderung ist dann das Aufgabenelement vom Typ `question`. Meist wird pro Aufgabenbeschreibung nur eine Frage oder Aufforderung gestellt, sodass es sehr verständlich ist, dies in einem Aufgabenelement als Aufgabenstellung zusammenfassen zu wollen. Allerdings gibt es auch Aufgaben, welche eine allgemeine Beschreibung (`constraint`) haben und anschließend mehrere Unteraufgaben, welche dann die Frage oder Aufforderung (`question`) enthalten. Folglich kann eine Differenzierung zwischen diesen beiden Typen weiterhin als sinnvoll erachtet werden.

*Nicht informatische Übungsblätter*

In einer weiteren Antwort aus dem Freitextfeld wurden die Typen `init`, `explanation` und `example` als ähnlich empfunden. Als Verbesserung wurde hier eine „feinere Unterteilung [der Datenstruktur je nach Anwendungsfall] in mathematisches Übungsblatt und informatisches Übungsblatt“ vorgeschlagen. Vermutlich wurde sich hier auf die Definitionen bezogen, da die Definitionen der beiden Typen `init` und `example` sich auf Code-Stücke beziehen und die Definition von `explanation` als Beispiel „Formeln“ nennt. Die entworfene Datenstruktur setzt den Fokus jedoch vorerst nur auf Übungsblätter, welche einen Bezug zur Informatik oder zum Programmieren haben. Das heißt rein mathematische Übungsblätter werden nicht explizit unterstützt, jedoch lassen sich einige vermutlich trotzdem mit der Datenstruktur klassifizieren.

*Erreichbare Punkte zu Aufgabe*

Ein weiterer Vorschlag für die Datenstruktur ist die Erfassung der zu erreichenden Punkte bei der Aufgabe. Dies ist eine sinnvolle Ergänzung, um auch Aufgaben zu unterstützen, welche keine Aufgabenelemente zur automatischen Korrektur enthalten. Dieser Vorschlag passt auch gut zu dem Vorschlag, die maximal erreichbaren Punkte automatisch aus den Aufgabentiteln zu entfernen, wenn die Aufgabenstruktur automatisch erkannt wird. Im gleichen Schritt könnten die Punkte dann an der passenden Stelle bei der Aufgabe in der Struktur gespeichert werden.
Auf die Speicherung der erreichbaren Teilpunkte bei den jeweiligen Aufgabenelementen vom Typ `grader` soll trotzdem nicht verzichtet werden, da diese auf konkrete Fehler in der Abgabe hinweisen können. Dadurch können dann bei der Generierung von Feedback die `grader`-Aufgabenelemente ignoriert werden, bei denen die volle Punktzahl erreicht wurde und nur welche mit Teilpunkten müssen beachtet werden.

### 5.2.4 Andere Anwendungsbereiche

*Unterschiedliche Darstellung der Elemente*

Die Teilnehmenden wurden am Ende der Evaluation gefragt, in welchen weiteren Bereichen sie die Klassifikation hilfreich finden könnten. Hierbei wurde vorgeschlagen, dass die Klassifizierung bei der Generierung von PDFs aus den *Jupyter-Notebook*-Dokumenten zur unterschiedlichen Formatierung genutzt werden kann. Von einer anderen Person kam ein ähnlicher Vorschlag, der sich auf die Darstellung in Lernmanagementsystemen bezieht. Durch die Klassifizierung können vorerst nur die Aufgabenstellung für die Lernenden freigeschaltet werden und Schritt für Schritt weitere Hinweise eingeblendet werden.

*Generierung von Aufgaben*

Ein weiterer Vorschlag von zwei Teilnehmenden ist, die Datenstruktur für eine Generierung von Aufgaben zu verwenden. Das bedeutet, dass die Struktur vorgegeben ist und dafür dann Inhalte generiert wird. Dies hat den Vorteil, dass die generierte Aufgabe anschließend auch schon direkt klassifiziert ist.

### 5.2.5 Vergleich der klassifizierten Jupyter-Notebook-Dokumente

Für die Evaluation wird auch die während der Umfrage vorgenommene Klassifizierung der Teilnehmenden des Beispiel-Dokumentes untersucht. Der Ausschnitt des Beispiel-Dokumentes, welcher klassifiziert werden sollte, ist in Anhang B zu sehen. Dort sind auch die Zellennummern angegeben. Da das Übungsblatt sowohl eine englische Version als auch eine deutsche Version enthält, sind die Zellen 20 und 22 sowie 21 und 23 inhaltlich identisch und wurden dementsprechend gleich klassifiziert. Eine Übersicht der gewählten Typen je nach Zelle ist in Abbildung 5.9 zu sehen.

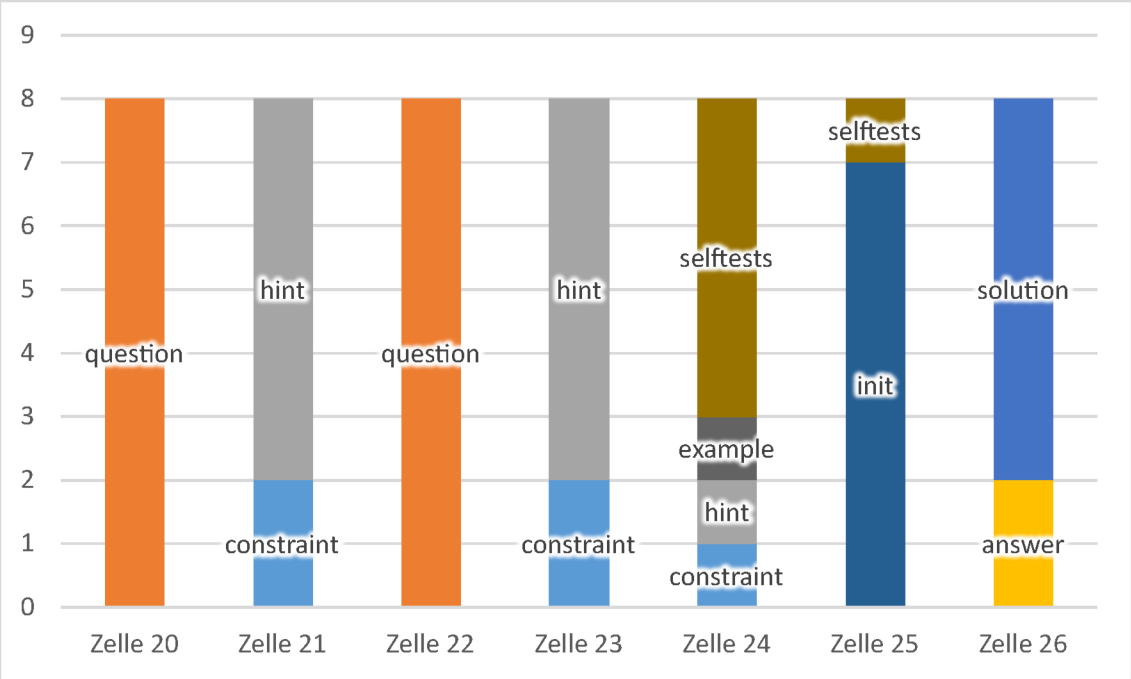



**Abbildung 5.9:** Übersicht über die von den Teilnehmenden gewählten Typen im Beispiel-Dokument, aufgeteilt nach Zellen. Die Zellen 20 und 22 sowie 21 und 23, waren inhaltlich identisch jedoch einmal auf Deutsch und einmal auf Englisch vorhanden.

Es wird deutlich, dass sich die Teilnehmenden bei einigen Zellen sehr einig sind, welcher Typ am besten passt und bei anderen weniger. Die Aufgabenstellung wurde von allen Teilnehmenden als solche identifiziert. Person P1 gab in einem Freitextfeld an, dass ihr die Klassifizierung des „‚Note' / ‚Hinweis' Feld[s]" (P1) (Zelle 21 und 23) schwerfiel und sie sich nicht zwischen `constraint` und `hint` entscheiden konnte. Dies spiegelt auch gut das Ergebnis der anderen Teilnehmenden wider.

Weitere unterschiedliche Angaben wurden für die Zelle 24 gemacht. Diese ist zwar beschriftet mit dem Titel „Als Hilfe:", jedoch wurde dort Code zur Verfügung gestellt mit dem die Antwort der Lernenden selbstständig überprüft werden kann: „Stellen Sie sicher, das [sic] der folgende Code mit ihrer Implementation funktioniert. [...]". Dies passt vom Inhalt zum Typ `selftests` und dies wurde auch von der Mehrheit der Befragten so wahrgenommen.

In Zelle 26 befindet sich die Lösung der Aufgabe. Offenbar haben zwei Personen angenommen, dass es sich um ein bearbeitetes *Notebook*-Dokument handelt, das bereits zurückgesendet wurde. Es handelte sich jedoch um ein *Notebook*-Dokument mit der Musterlösung, dies wurde allerdings in der Umfrage nicht kommuniziert.

## 5.3 Datenstruktur

Die vorläufige Datenstruktur aus Kapitel 3 ist anhand der Evaluationsergebnisse überarbeitet worden und wird in diesem Kapitel präsentiert. Dafür werden die vorgenommenen Änderungen erläutert und diskutiert. Anschließend wird das überarbeitete XML-Schema der Datenstruktur vorgestellt.

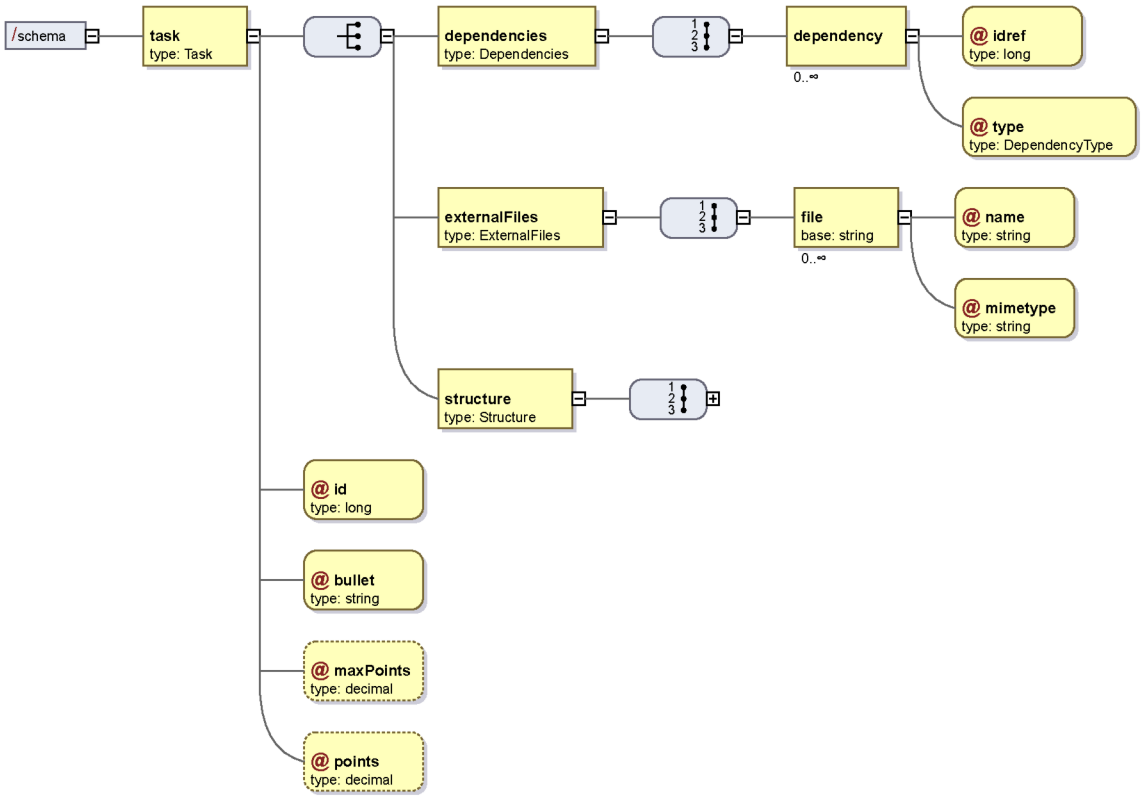



**Abbildung 5.10:** Grafische Darstellung eines Ausschnittes des überarbeiteten XML-Schemas für die Datenstruktur.

### 5.3.1 Änderungen an der Datenstruktur

*Maximal erreichbare Punkte*

Das `task`-Element zur Repräsentation einer Aufgabe in der Datenstruktur wurde um Attribute für die maximal erreichbaren Punkte und die tatsächlich erreichten Punkte ergänzt. Dies hat den Vorteil, dass auch extern bewertete Aufgaben ohne `grader`-Elemente gefiltert werden können, je nachdem, ob die jeweilige Aufgabe vollständig erfüllt wurde oder eben nicht. Für die automatische Feedback-Generierung fehlt dann leider ein Hinweis auf den Grund, warum nicht die volle Punktzahl erreicht wurde, da dieser nicht aus den `grader`-Elementen abgeleitet werden kann.

*Eindeutige IDs*

Durch die Einführung der genauen Angabe, von welcher (Teil-)Aufgabe eine (Teil-)Aufgabe abhängig ist, ist eine präzise Referenzierung von Aufgaben erforderlich. Es wurde entschieden, das Attribut `bullet` der `task`-Elemente nicht zu verwenden. Dadurch hätte eine Aufgabe über einen Pfad (z.B. „1/a“) spezifiziert werden können. Dies bringt jedoch Einschränkungen bei den Zeichen für die Aufgabentitel mit sich und zusätzlich müssten Aufgabentitel eindeutig sein. Deswegen wurde für jede Aufgabe ein Attribut mit dem Namen `id` eingeführt, welche in der gesamten Struktur eindeutig sein muss.

*Abhängigkeit von Aufgaben*

Das komplette XML-Schema im XSD-Format befindet sich in Anhang E. In Abbildung 5.10 sind die neuen Attribute für das `task`-Element dargestellt. Ebenso ist hier auch abgebildet, wie die Speicherung der Abhängigkeit zu anderen Aufgaben erfolgen soll. Hierzu wurde ein neues Kind-Element mit dem Tagnamen `dependencies` zu

jedem `task` hinzugefügt. Innerhalb des `dependencies`-Tags kann in einer beliebig langen Liste angegeben werden, von welchen Aufgaben die aktuelle Aufgabe abhängt. Dafür werden die neu eingeführten `dependency`-Tags verwendet.
Ein `dependency`-Tag muss die zwei Attribute `idref` und `type` enthalten. In `idref` wird die ID der Aufgabe gespeichert, von der eine Abhängigkeit besteht. Für das Attribut `type` werden aktuell die beiden Werte „description“ und „result“ unterstützt. Diese geben wie in der ursprünglichen Datenstruktur an, ob eine Abhängigkeit nur von der Aufgabenstellung besteht oder auch noch zusätzlich von der Antwort. Letzteres ist zum Beispiel der Fall, wenn Funktionen aus vorherigen Teilaufgaben für eine Aufgabe benötigt werden.
Vorgesehen ist hier die Referenzierung von Teilaufgaben, welche ebenfalls Kinder der gleichen übergeordneten Aufgabe sind und vor dieser Aufgabe bearbeitet werden sollen. Im Falle von Aufgaben, welche keine übergeordnete Aufgabe haben und sich somit auf erster Ebene in einem Übungsblatt befinden, sollen alle vorherigen Aufgaben auf erster Ebene referenziert werden können.

*Reduzierung der `hint`-Varianten*

Eine weitere Änderung ist die Reduzierung der Typen für die Aufgabenelemente, da einige Typen zu dem Typ `hint` zusammengefasst wurden. Dies ist in Abbildung 5.11 zu sehen. Elemente des Typs `hint` können nun ein weiteres optionales Attribut `type` besitzen. Mit diesem kann die Art des Hinweises weiter spezifiziert werden. Dabei sind die Werte „constraint“, „explanation“ und „example“ für das Attribut erlaubt. Für allgemeine Hinweise zur Beantwortung der Aufgabe, welche keine weitere Spezifizierung erfordern, wird das Attribut weggelassen.

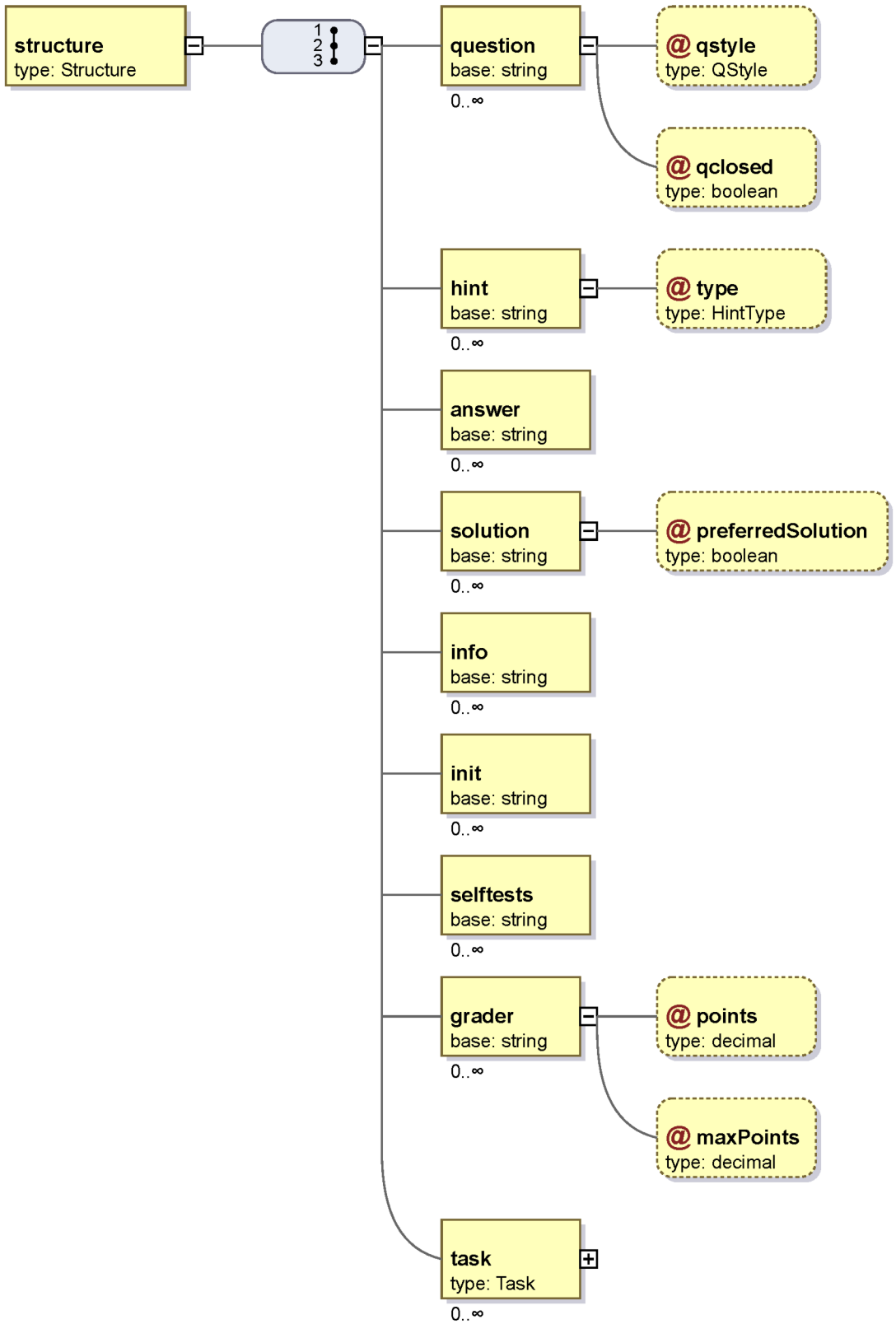



**Abbildung 5.11:** Ausschnitt aus dem überarbeiteten XML-Schema der Datenstruktur zur Darstellung der verschiedenen Aufgabenelementtypen mit ihren Attributen. Der `HintType` steht für die Optionen „constraint“, „explanation“ oder „example“. Der Typ `QStyle` steht für die Optionen „multiple-choice“, „write-code“, „explain-code“, „draw“ und „parsons-problem“.

# Kapitel 6 Diskussion

In diesem Kapitel werden verschiedene Entscheidungen und Ergebnisse der Arbeit diskutiert. Dabei wird auf die Analyse der *Notebook*-Dokumente eingegangen und inwiefern die Ergebnisse der verwandten Arbeiten einen Einfluss auf die Datenstruktur haben. Es wird erörtert, wie das Plugin an die überarbeitete Datenstruktur angepasst werden kann und welche Unterschiede zu weiteren Plugins in anderen Umgebungen als *JupyterLab* auftreten können. Abschließend werden Limitationen der Evaluation dargelegt und welche Erkenntnisse aus dieser gezogen werden können.

## 6.1 Analyse der Notebook-Dokumente

*Anzahl Notebook-Dokumente*

Auf Grund begrenzter Kapazitäten im Rahmen dieser Masterarbeit sind nur die Übungsblätter von fünf Kursen analysiert worden. Die fünf Kurse haben alle einen Bezug zur Informatik und in allen müssen Programmieraufgaben gelöst werden. Da es zwischen den Kursen viele Gemeinsamkeiten in den vorhandenen Aufgabenelementtypen gibt, kann der Datensatz für ein Proof-of-Concept der Datenstruktur genutzt werden, obwohl es sich um einen kleinen Datensatz handelt.

*Klassifizierung nur von einer Person*

Die *Jupyter-Notebook*-Dokumente wurden für diese Arbeit nur von einer Person klassifiziert. Eine andere Person kann bei gleicher Datengrundlage zu leicht anderen Ergebnissen kommen. Dies ist bei der Auswertung der *Notebook*-Dokumente aus der Evaluation zu sehen, in welcher die Teilnehmenden die Klassifizierung leicht unterschiedlich vornahmen. Für aussagekräftigere Ergebnisse sollte ein größerer Datensatz an Übungsblättern aus mehreren Kursen von verschiedenen Personen nach möglichen Aufgabenelementtypen durchsucht und klassifiziert werden. Die bisher erarbeitete Datenstruktur bietet jedoch einen guten Ausgangspunkt für eine solche Analyse.

## 6.2 Datenstruktur

Die erarbeiteten Typen für die Aufgabenelemente basieren zu großen Teilen auf den analysierten *Jupyter-Notebook*-Dokumenten und den Interviews. Dies liegt vor allem daran, dass bisherige Forschungsarbeiten den Fokus auf eine Klassifizierung von Aufgaben als Ganzes gelegt haben. Auch von diesen Klassifizierungen wurden einige Teile in die Datenstruktur integriert.

*Integration bestehender Klassifizierungen*

Die Klassifizierung, ob es sich um eine geschlossene Fragestellung handelt, kann bei dem `question`-Typ gespeichert werden [15, 62, 40]. Außerdem kann bei dem `question`-Typ die Art der Frage angegeben werden. Dafür stehen die Arten „Multiple choice“, „Write code“, „Explain code“, „Draw“ und „Parsons problem“ zur Auswahl. Zum einen wird hier etwas das erwartete Format der Antwort integriert [59]. Zum anderen geht es hier hauptsächlich um die Klassifizierung nach dem Skill-Level [62, 13, 15, 59, 3]. Da es keine eindeutige Liste an Skill-Leveln gibt, wurde sich hier auf eine kompakte Auswahl beschränkt, welche auch in den analysierten Übungsblättern vorkam. Dies erleichtert auch die Auswahl bei der Klassifizierung der Übungsblätter. Die Liste kann in dem Plugin jedoch leicht angepasst und erweitert werden.
Die maximal erreichbaren Punkte einer Aufgabe werden ebenfalls in der Datenstruktur gespeichert, jedoch nicht um die Schwierigkeit der Aufgabe festzustellen [62]. Die Speicherung zielt darauf ab, in Kombination mit den erzielten Punkten zu identifizieren, ob bei einer abgegebenen Antwort ein erläuterndes Feedback erforderlich ist oder die Aufgabe korrekt bearbeitet wurde.

*Andere Fachbereiche*

In dieser Arbeit wurde der Fokus auf digitale Übungsblätter im Bereich der Programmierung gelegt. Das bedeutet, dass sich die Datenstruktur nicht unbedingt auf andere Übungsblätter übertragen lässt. Jedoch kann davon ausgegangen werden, dass die grundlegende Struktur einer Übungsaufgabe auch in anderen Bereichen angewandt werden kann.

*Relevanz der Typen*

Bisher wurden die Aufgabenelementtypen mit den dazugehörigen Eigenschaften anhand der Übungsblätter gesammelt. Es fehlt jedoch bisher eine Analyse, welche Informationen tatsächlich für die Generierung von Feedback hilfreich sind. Wenn die Generierung von Feedback durch LLMs stattfinden soll, dann gibt es einige Studien zum Prompt Engineering, jedoch werden bei keiner nur Elemente der Aufgabe als Input gegeben [57, 17]. Eine Ausnahme bildet die Musterlösung, welche in den Studien als eigener Teil behandelt wird [57, 17, 42].
Durch die Datenstruktur ist es jedoch möglich zu untersuchen, welche Aufgabenelemente für eine Feedback-Generierung relevant sind. Ein solches Prompt Engineering ist jedoch nicht im Rahmen dieser Masterarbeit vorgesehen.

## 6.3 Plugin

*Anpassbarkeit der Typen*

Die Anforderungen an das Plugin werden vollständig erfüllt. Zudem wird das Plugin als nutzungsfreundlich erachtet. Die wichtigste Anforderung an das Plugin ist dabei die Unterstützung der entwickelten Datenstruktur. Hier wurde bei der Entwicklung auch Wert daraufgelegt, dass eine Änderung an den Aufgabenelementtypen oder den dazugehörigen Eigenschaften schnell in das Plugin übersetzt werden können.

*Neue Angaben bei einer Aufgabe*

Jedoch wurden bei der Überarbeitung der Datenstruktur auch Änderungen an den Eigenschaften einer Aufgabe vorgenommen und wie Abhängigkeiten zwischen (Teil-)Aufgaben angegeben werden können. Dies führt zu größeren Änderungen an dem Plugin, welche nach der Evaluation nicht mehr umgesetzt wurden. Das Plugin müsste dafür nicht nur die neue Datenstruktur unterstützen, sondern teilweise auch eine neue Oberfläche zur Verfügung stellen, in welcher die weiteren Angaben gemacht werden können.

*Aufteilen in Aufgabenelemente*

Das Plugin wurde für eine *JupyterLab*-Umgebung entwickelt und ist auch nur für die Klassifizierung von *Jupyter-Notebook*-Dokumenten geeignet. Viele Aufgaben in *Notebook*-Dokumente sind jedoch oft schon in verschiedene Zellen aufgeteilt, sodass die Nutzenden hier keine eigenständige Aufteilung vornehmen müssen. Dadurch wurde auch nur untersucht, wie schwierig das Zuweisen von Kategorien zu den Zellen ist. Eine Untersuchung der Schwierigkeit vom Aufteilen von Aufgaben in die Kategorien wurde nicht betrachtet. Es kann jedoch angenommen werden, dass es sich hierbei um einen Sonderfall handelt. Bei der Darstellung von digitalen Übungsblättern auf anderen Plattformen muss dieser Aspekt vermutlich berücksichtigt werden.

## 6.4 Evaluation

*Qualitative Studie*

Die Evaluation wurde mit nur acht Teilnehmenden durchgeführt. Für eine quantitative Auswertung der Ergebnisse müssen jedoch deutlich mehr Personen an der Evaluation teilnehmen. Durch die ganzen Freitext-Antworten lässt sich die Studie jedoch qualitativ auswerten.

*Rolle der Teilnehmenden*

Die Zielgruppe des Plugins sind vor allem die Lehrenden, welche die Übungsblätter erstellen. Die Hälfte der Befragten hat schon mal ein Übungsblatt erstellt. Für die Auswertung der Nutzungsfreundlichkeit des Plugins ist es nicht relevant, ob die Zielgruppe in der Gruppe der Testpersonen stark vertreten ist. Allerdings kann die Klassifizierung des Übungsblattes aus Sicht der Erstellenden anders ausfallen.

*Beispiel-Übungsblatt*

Da die Evaluation nur eine kleine Zahl an Teilnehmenden hatte, wurde hier versucht, die Ergebnisse so vergleichbar wie möglich zu halten. Aus diesem Grund sollten die Teilnehmenden die Änderungen alle an dem gleichen Übungsblatt vornehmen. Die Teilnehmenden kannten sich jedoch nicht alle mit dem Inhalt des Übungsblattes aus. Dies macht eine Klassifizierung des Inhalts unterschiedlich schwer, je nachdem wie vertraut die Befragten mit der Materie des Übungsblattes sind.

Eine Auswertung mit Personen, die selbst Übungsblätter erstellen und diese dann klassifizieren, bringt genauere Einblicke in das Verständnis der einzelnen Kategorien. Um eine quantitative Auswertung durchzuführen, sollten auch mehrere Übungsblätter pro Kurs von den Personen klassifiziert werden, wenn möglich auch mehrere Lehrende pro Übungsblatt.

*SUS-Score*

Auch der *System Usability Scale (SUS) Score* wird durch eine größere Stichprobe an Befragten aussagekräftiger. Das bisher erfasste positive Ergebnis gibt trotzdem schon mal eine Tendenz an.

*Aufsplittung der Gruppen*

Eine Auswertung nach den verschiedenen Personengruppen ist zwar möglich, jedoch ist die Anzahl der Befragten je Gruppe dadurch sehr gering. Es wurden die in Tabelle 6.1 beschriebenen Aufteilungen untersucht. Dadurch ließen sich jedoch keine großen Unterschiede in den Antworten von je zwei Gruppen finden.

*Kein reales Anwendungsszenario*

Das Plugin wurde von den Teilnehmenden im Rahmen der Umfrage getestet. Die Teilnehmenden wurden dafür schrittweise mit einer Aufgabenliste durch das Plugin geführt. Dadurch wurden die wichtigsten Funktionen von allen getestet. Allerdings handelt es sich hierbei nicht um ein reales Anwendungsszenario. In einem realen Szenario sind nicht zwangsläufig alle Funktionen erforderlich. Da das Ergebnis für die Korrektur der eigenen Lehrveranstaltung genutzt werden kann, ist die Motivation, die Klassifizierung ordentlich zu machen, vermutlich höher.

**Tabelle 6.1:** Überprüfte Aufsplittungen der Teilnehmenden der Evaluation in jeweils zwei Gruppen für die Analyse der abgegebenen Antworten.

| Kriterium | Gruppe 1 | Gruppe 2 |
|---|---|---|
| Alter | Unter 30 Jahre | Über 30 Jahre |
| Rolle | Dozent*in, Tutor*in oder „Übungsleiter“ (P8) | Weder Dozent*in, Tutor*in noch „Übungsleiter“ (P8) |
| JupyterLab-Erfahrung | Keine | Mind. Jupyter Notebook |
| JupyterLab-Erfahrung | Maximal Jupyter Notebook | Eine Weile her - täglich |
| JupyterLab-Erfahrung | Keine - Eine Weile her | Gelegentlich - täglich |
| JupyterLab-Erfahrung | Keine - Gelegentlich | Wöchentlich - täglich |
| JupyterLab-Erfahrung | Keine - Wöchentlich | Nutzt täglich JupyterLab |
| Übungsblatt | Erstellt | Bisher noch nicht erstellt |
| Übungsblatt | Korrigiert | Bisher noch nicht korrigiert |
| Übungsblatt | Verwendung der Korrekturergebnisse | Keine Verwendung |

*Formulierung im Fragebogen*

Der Fragebogen für die Evaluation konnte von den Befragten fast ohne Rückfragen beantwortet werden. Nur die Frage nach den anderen Bereichen für den Einsatz der Datenstruktur bedarf einer zusätzlichen Erklärung. Die erhobenen Daten können gut für eine Auswertung genutzt werden. Jedoch ist die Frage „Wenn vorhanden, habe ich die weiteren Felder für diesen Typ ausgefüllt“ überflüssig, da die bearbeiteten *Notebook*-Dokumente dafür betrachtet werden können.
Wenn den Befragten für die Klassifizierung ein Übungsblatt zur Verfügung gestellt wird, sollte Ihnen mitgeteilt werden, ob es sich um ein Übungsblatt mit den Musterlösungen oder um ein von den Lernenden bearbeitetes Übungsblatt handelt. Des Weiteren musste die Frage nach anderen Anwendungsfeldern für die Datenstruktur für die meisten Befragten mündlich erläutert werden. Für zukünftige Evaluationen empfiehlt es sich, diese Frage genauer zu formulieren, um Missverständnisse oder Rückfragen zu vermeiden.

# Kapitel 7 Zusammenfassung

In dieser Arbeit wurde eine Datenstruktur entworfen, mit welcher Übungsaufgaben-Elemente klassifiziert werden können. Zusätzlich wurde ein *JupyterLab*-Plugin entwickelt, um die Klassifizierung von *Jupyter-Notebook*-Dokumenten zu vereinfachen. Anschließend wurde beides mittels einer Umfrage ausgewertet. In den folgenden Abschnitten wird auf die Forschungsfragen aus Abschnitt 1.2 eingegangen.

*Kontext für Übungsaufgaben*

Bei der Feedback-Generierung mit *Large Language Models* (LLMs) können strukturierte Prompts bei einer Anfrage hilfreich sein [57, 17]. Für die Feedback-Generierung ist die Idee, Übungsaufgaben mit einem Kontext zu versehen, welcher den Aufgaben eine Struktur gibt. Dadurch können bei der Erstellung des Prompts weniger relevante Elemente ausgeschlossen werden und der Prompt erhält eine klare Struktur (RQ1).

*Datenstruktur*

Zusätzlich zu einer Klassifizierung von Aufgabenelementen ist eine Erfassung der Struktur der Aufgaben und Teilaufgaben sinnvoll. Durch diese Struktur können Abhängigkeiten von vorherigen (Teil-)Aufgaben angegeben werden. Zusätzlich wird eine Abhängigkeit zu Elementen aus einer übergeordneten Aufgabe impliziert (RQ2).
Es wurden acht Typen herausgearbeitet, in die sich die Aufgabenelemente von Übungsblättern im Bereich der Programmierung einordnen lassen (RQ2.1). Einzelne Typen lassen sich mit zusätzlichen Angaben noch weiter spezifizieren (RQ2.2). Die Typen wurden mit Hilfe von digitalen Übungsblättern aus verschiedenen Kursen in Form von *Jupyter-Notebook*-Dokumenten erarbeitet. Anschließend wurden die Typen in einer Umfrage evaluiert. Im Rahmen dieser Masterarbeit wurde noch keine Analyse zur Nützlichkeit der erarbeiteten Typen und ihrer Eigenschaften in Bezug auf die Feedback-Generierung mit zum Beispiel LLMs durchgeführt.

*Plugin*

Um die Erstellung der Datenstruktur aus einem Übungsblatt zu unterstützen, wurde exemplarisch für die *JupyterLab*-Umgebung ein Plugin entwickelt. Dieses speichert während der Erstellung die Informationen innerhalb des *Jupyter-Notebook*-Dokumentes und kann die Daten anschließend in der erarbeiteten Datenstruktur als XML exportieren. Dies ermöglicht eine erhebliche Zeitersparnis, da das XML nicht per Hand geschrieben werden muss. Zudem können bestehende Übungsblätter mit diesen Informationen unkompliziert ergänzt werden (RQ3).

*Evaluation*

In der Evaluation wurde das Plugin als nützlich bewertet, jedoch wurden von den Teilnehmenden auch mehr erklärende Texte gewünscht, welche bei der ersten Verwendung helfen. Vor allem eine Erklärung der nur durch Symbole dargestellten Buttons wurde mehrfach angemerkt. Des Weiteren wurde die Datenstruktur teilweise als zu

komplex wahrgenommen, da es relativ viele Aufgabenelementtypen gab. Die Auswertung der Evaluation des verwendeten Übungsblattes zeigt, dass die befragten Personen für die einzelnen Elemente trotzdem eine ähnliche Klassifizierung vorgenommen haben. Die Anzahl der Typen wurde jedoch in einer nach der Evaluation überarbeiteten Version der Datenstruktur weiter reduziert. Die reduzierte Liste der Typen ist in Abbildung 5.11 als Auszug aus dem XML-Schema der überarbeiteten Datenstruktur dargestellt.

# Kapitel 8 Ausblick

Das letzte Kapitel behandelt weitere Verbesserungen des Plugins, welche sich in der Evaluation ergeben haben und die Anpassungen, welche vorgenommen werden müssen, um die überarbeitete Datenstruktur zu unterstützen. Außerdem wird darauf hingewiesen, dass größere Studien für eine quantifizierbare Auswertung der Datenstruktur nötig sind. Ebenso wird darauf hingewiesen, dass Studien fehlen, die analysieren, wie gut Feedback mit der entworfenen Datenstruktur generiert werden kann.

## 8.1 Plugin

Einige der Verbesserungsvorschläge für das Plugin wurden nach der Evaluation bereits umgesetzt. Jedoch wurden noch nicht alle Vorschläge der Befragten übernommen.

*Drag-and-Drop von Dateien*

Eine Person hat versucht, eine Datei aus dem Dateimanager per Drag-and-Drop auf die Aufgabe in der Struktur zu ziehen. Dies ist eine sehr intuitive Art, um eine Datei mit einer Aufgabe zu verknüpfen. Hierzu müsste jedoch erst überprüft werden, wie dies in *JupyterLab* umsetzbar ist, da durch das modulare Design auch der Dateimanager durch eine Erweiterung realisiert wird. In *JupyterLab* ist die Kommunikation zwischen Erweiterungen nur durch definierte Schnittstellen möglich.

*Eigenschaften einer Aufgabe*

Durch die Veränderung der Datenstruktur nach der Evaluation sind weitere Eigenschaften für ein `task`-Element hinzugekommen. Die Änderungen an den Aufgabenelementtypen ließ sich in dem Plugin einfach umsetzen, da bei der Implementation darauf geachtet wurde. Jedoch war zum Zeitpunkt der Planung der Implementation nicht vorgesehen, dass sich viel an dem Aufgabenelement verändert. Die Änderungen, welche sich in der Evaluation ergeben haben, erfordern jedoch nicht nur Änderungen an der Datenstruktur, sondern auch an der Oberfläche zur Erstellung der Datenstruktur.

*Bearbeiten eines Aufgabenelementes*

Ein weiteres Problem ist, dass die Oberfläche zum Bearbeiten eines Aufgabenelementes nicht von allen Befragten direkt gesehen wurde. Eine Person hat auch erwartet, dass man eine Änderung ähnlich wie die Bearbeitung an einer Aufgabe vornimmt, in dem dort auch neben dem Aufgabenelement Symbole erscheinen. Es wurde auch versucht, einen Änderungsvorgang mit einem Rechtsklick anzustoßen. Hier sollte in einer zukünftigen Version des Plugins darauf geachtet werden, wie die Oberfläche zur Bearbeitung eines Aufgabenelementes besser sichtbar gemacht werden kann.

*Erklärung der Typen*

Die verwendeten Typen der Aufgabenelemente werden in dem Plugin kurz erläutert, sobald diese ausgewählt werden. Hier wurde jedoch erwähnt, dass eine Übersicht mit allen Erklärungen die Benutzung noch weiter vereinfachen würde. Dies würde sich zum Beispiel durch eine größere Auswahlliste realisieren, welche nicht nur die Namen der Typen enthält. Pro Eintrag könnten dann der Namen als Überschrift formatiert werden und noch die kurze Beschreibung direkt darunter.

*Automatische Erkennung des Typs*

Um den Zeitaufwand bei der Klassifizierung weiter zu minimieren, könnte man überprüfen, ob ein LLM die Klassifizierung der einzelnen Elemente unterstützen kann. Dieses müsste vermutlich mit ein paar Beispielen angelernt werden. Dadurch könnte nicht nur die Struktur eines Blattes automatisch erkannt werden, sondern soweit möglich auch eine erste Klassifizierung vorgenommen werden. Diese müsste dann trotzdem noch manuell von den Nutzenden überprüft werden. Hier sollte dann auch überprüft werden, ob dies eine Zeitersparnis ist und weniger Fehler als bei einer manuellen Klassifizierung entstehen.

*Integration in andere Systeme*

Auch sollten weitere Implementationen in andere Systeme vorgenommen werden, um die Klassifizierung von digitalen Übungsblättern dort zu untersuchen. Ein großer Unterschied hierbei wird es vermutlich sein, dass die Aufgaben noch nicht in einzelne Aufgabenelemente unterteilt sind. Eine Aufteilung von bestehenden Aufgaben in Aufgabenelemente wurde in dieser Arbeit nicht evaluiert.

## 8.2 Datenstruktur

Die Ergebnisse der Evaluation wurden in die Datenstruktur eingearbeitet, welche anschließend in dieser Arbeit präsentiert wurde. Jedoch sind die Ergebnisse der Studie nicht quantitativ auswertbar. Die Auswirkungen sollten auch mit weiteren Studien überprüft werden.

*Größere Studie*

Für quantifizierbare Aussagen zu der Datenstruktur müssen mehr Personen befragt werden. Auch sollten in einer größeren Studie unterschiedliche Übungsblätter klassifiziert werden, idealerweise von den Personen, welche die Übungsblätter erstellt haben. Dadurch ist gewährleistet, dass sie mit der Materie vertraut sind und eine sinnvolle Klassifizierung vornehmen können.

In einer größeren Studie kann dann auch darauf geachtet werden, dass mehrere Personen mit der Materie eines Übungsblattes vertraut sind. Dann sollte ein Übungsblatt jeweils von mehreren Personen klassifiziert werden, um anschließend zu analysieren, welche Aufgabenelementtypen oft eindeutig zugeordnet werden konnten und bei welchen es eher zu Unstimmigkeiten kommt.

*Praktische Relevanz der Typen*

Eine Analyse zur Feedback-Generierung basierend auf dieser Datenstruktur wurde noch nicht durchgeführt, da zum Beispiel auch noch keine größeren Mengen von Übungsblättern in der Datenstruktur klassifiziert wurden. Im Rahmen einer zukünftigen Studie sollte die praktische Relevanz der einzelnen Aufgabenelementtypen für die Generierung von Feedback untersucht werden. Dafür müssen verschiedene Aufgabenelemente für eine Feedback-Generierung kombiniert und die Ergebnisse manuell bewertet werden.

*Kürzere Prompts für LLMs*

Dies kann zum Beispiel mit Hilfe von *Large Language Models (LLMs)* geschehen. Die Länge des Prompts für ein LLM kann durch die Auswahl von relevanten Aufgabenelementen anhand ihres Typs und der weiteren Informationen deutlich verkürzt werden im Vergleich zu einem Prompt mit der vollständigen Aufgabenbeschreibung. Des Weiteren besteht nun die Möglichkeit, Prompts für einzelne Teilaufgaben unter Nutzung der relevanten Informationen zu erstellen, ohne dabei eine komplette Aufgabe mit allen Teilaufgaben nutzen zu müssen.

*Spezifischere Prompts für LLMs*

Zusätzlich kann untersucht werden, ob die Aufgabenelementtypen oder die zusätzlichen Informationen auch bei Benennung im Prompt einen Vorteil bieten. Dies kann zum Beispiel wie in den Arbeiten von Roest et al. und Hellas et al. durch Zwischenüberschriften gemacht werden, welche dann das jeweilige Aufgabenelement bei der Verwendung im Prompt beschreiben [57, 17]. Es ist ebenfalls denkbar, dass ein Fine-Tuning für ein LLM anhand von klassifizierten Aufgaben und Feedback hilfreich ist. Dadurch kann dem LLM beigebracht werden, dass weniger Fokus auf Elemente gelegt werden soll, die als Musterlösung gekennzeichnet sind.

*Generierung von Aufgaben*

In der Evaluation wurde von zwei Personen vorgeschlagen, dass die Datenstruktur auch zur Generierung von Aufgaben genutzt werden könnte. Da bei der Generierung bereits die Klassifizierung vorgegeben ist, muss diese anschließend nicht mehr erfasst werden. Dadurch entstehen direkt klassifizierte Aufgaben, welche so für die Feedback-Generierung verwendet werden können.

*Integration in ProFormA*

Wenn sich in zukünftigen Arbeiten herausstellt, dass die Feedback-Generierung durch die Datenstruktur verbessert werden kann, dann wäre so eine Klassifizierung langfristig in dem *ProFormA*-Standard hilfreich. *ProFormA* ist primär für den Austausch von Aufgaben über verschiedene Plattformen hinweg gedacht und mit Version 2.0 wurde auch die Kommunikation mit Bewertungssystemen definiert. Die Datenstruktur kann von den verschiedenen Systemen, welche *ProFormA* unterstützen, auch für die Darstellung der verschiedenen Aufgabenelemente verwendet werden, wie zum Beispiel dem späteren Freischalten von Hinweisen.
Eine Integration der Aufgabenelemente ist in *ProFormA* einfach möglich, da sie anstelle der ursprünglichen Aufgabenstellung platziert werden können. Komplizierter ist es, Aufgaben mit Teilaufgaben darzustellen, da *ProFormA* nicht für die Abbildung von Aufgabenstrukturen konzipiert wurde. Ähnlich komplex ist es mit der Repräsentation von aufgabenübergreifenden Aufgabenelementen, welche für alle Aufgaben eines Übungsblattes relevant sind.

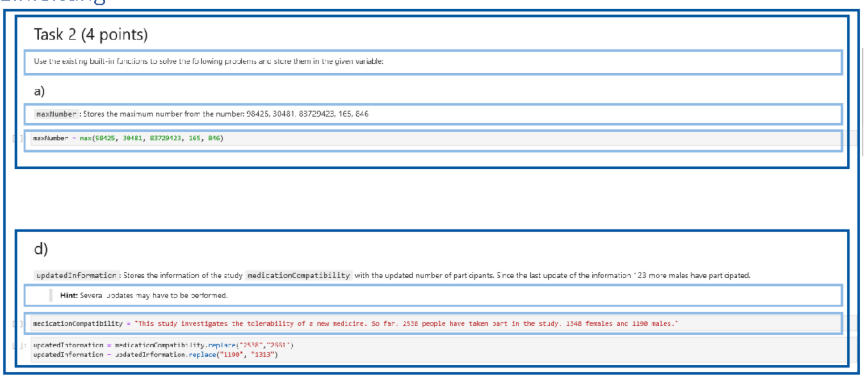
## Acknowledgement

Mein Dank geht an den Lehrstuhl für Informationssysteme und Datenbanken, den Lehrstuhl für Methodik und Theorie computerbasierter Geistes- und Sozialwissenschaften, den Lehrstuhl für Data Science im Maschinenbau und das Lehr- und Forschungsgebiet Lerntechnologien der RWTH Aachen für die Bereitstellung der *Jupyter-Notebook*-Dokumente, welche für die Analyse in dieser Arbeit verwendet wurden.
In dieser Arbeit wurde für die Verständlichkeit und Grammatik einzelner Sätze die Unterstützung von *DeepL Write*[1] in Anspruch genommen. Die von dem Tool generierten Texte wurden von mir persönlich überprüft und angepasst, bevor sie in die Arbeit übernommen wurden.


[1] https://www.deepl.com/write

# Anhang A Interviewfragen

## Einleitung




Datenstruktur zur Repräsentation der Struktur eines Arbeitsblattes

Ziel: Kontext für Übungsaufgaben geben, sodass Feedback auf Abgaben automatisch generiert werden kann (z.B. mit LLM) + Plattformunabhängige Auswertung möglich

## Wer wird befragt

### Wie viele Teilnehmende gibt es ungefähr in dem Kurs?

### Welche Programmiersprache(n) wird/werden in dem Kurs benötigt?

### Liegt der Fokus des Kurses auf dem Erlernen neuer Programmiersprachen oder wird ein grundlegendes Programmierverständnis vorausgesetzt?

### Für welche Studiengänge (+ Bachelor/Master) wird der Kurs hauptsächlich angeboten?

## Wie sieht der aktuelle Ablauf für die Korrektur aus

### Wie wird die Korrektur der Abgaben aktuell durchgeführt? (manuell, halbautomatisch, automatisch)

- Wie viel zeitlicher Aufwand war es die (halb-)automatische Korrektur aufzusetzen (pro Übungsblatt)?
- Falls bekannt:
  - Wie viele gültige Antworten werden als falsch erkannt?
  - Wie viele falsche Antworten werden als richtige Lösung akzeptiert?

### Wie viel Zeit wird durchschnittlich für die Korrektur eines Arbeitsblattes benötigt?

### Wie viel Zeit dürfte das Klassifizieren der Teile eines Übungsblattes in Anspruch nehmen?

Hinweis: Potenziell kann das Schreiben der Testfälle damit wegfallen

## Feedback Qualität

### Was für eine Art Feedback wird bisher gegeben?

- Punktzahl für gesamtes Übungsblatt: (Ja/Nein)
- Welche Aufgaben richtig sind und welche nicht:
- Eine mögliche richtige Lösung:
- Hinweis, was an der aktuellen Antwort falsch war:
- Hinweis, wie die aktuelle Antwort verbessert / korrigiert werden kann:

## Anforderungen an die Erweiterung

### Wie weit wird das Notebook von den Studierenden während der Bearbeitung verändert?

1. Es wird nur in dafür vorgesehene Zellen geschrieben
2. Es werden weitere Zellen ergänzt
3. Es werden Zellen ergänzt / gelöscht / verschoben
4. Die Antworten stehen nicht mehr an der richtigen Stelle

### Wo werden die Übungsblätter bearbeitet?

- Auf zentralem Server (JupyterLab/Hub/...?), der den Studierenden zur Verwendung zur Verfügung gestellt wird
- Die Studierenden kümmern sich selbst um eine Lösung

### Existieren Musterlösungen und wenn ja, wo und wie sind diese gespeichert?

# Anhang B Ausschnitt aus dem Beispiel-Dokument für die Evaluation

20 b)

In the next step you are to define the exception `NicholasToFatException` which can be thrown if Santa Claus with his gift bag does not fit through a chimney. As an additional parameter the exception shall receive an information `missing_space` , which indicates by how many mm the Santa Claus is too thick for the current chimney. As soon as the exception is thrown, the following sentence is to be output: `For the given chimney, the Nicholas is {missing_space} mm to fat.`

21 > **Note:** Use the given constant `ERROR_MESSAGE_NICHOLAS_TO_FAT` to construct the error message to avoid point deductions due to errors in the output sentence.

---

22 Im nächsten Schritt sollen Sie die Exception `NicholasToFatException` definieren, die ausgelöst werden kann, wenn der Nikolaus mit seinem Geschenkesack nicht durch einen Schornstein passt. Als zusätzlichen Parameter soll die Exception eine Information `missing_space` erhalten, die angibt, um wie viele mm der Nikolaus zu dick für den aktuellen Schornstein ist. Sobald die Exception ausgelöst wurde, soll der folgende Satz ausgegeben werden: `For the given chimney, the Nicholas is {missing_space} mm to fat.`

23 > **Hinweis:** Benutzen Sie zur Konstruktion der Fehlermeldung die vorgegebene Konstante `ERROR_MESSAGE_NICHOLAS_TO_FAT` , um Punktabzüge durch Fehler im Ausgabesatz zu vermeiden.

---

24 For help: / Als Hilfe:
Make sure that the following code works with your implementation:
Stellen Sie sicher, das der folgende Code mit ihrer Implementation funktioniert.

```
exception = NicholasToFatException(5)
print(exception)
print(repr(exception))


# The following should be printed:
For the given chimney, the Nicholas is 5 mm to fat.
NicholasToFatException('The Nicholas is to fat.')
```

25 In [ ]:
```
ERROR_MESSAGE_NICHOLAS_TO_FAT = "For the given chimney, the Nicholas is {} mm to fat."
```

26 In [ ]:
```
class NicholasToFatException(Exception):
    def __init__(self, missing_space):
        Exception.__init__(self, "The Nicholas is to fat.")
        self.missing_space = missing_space

    def __str__(self):
        return ERROR_MESSAGE_NICHOLAS_TO_FAT.format(self.missing_space)

# the following only helps for understanding
exception = NicholasToFatException(5)
print(exception)
print(repr(exception))
```

# Anhang C Evaluationsfragebogen (de)



**Einverständniserklärung**

**Zweck und Inhalt der Studie**
In der Umfrage werden die Teilnehmenden nach ihrer Meinung zum JupyterLab Plugin „Classify Task Elements“ und der zugrundeliegenden Datenstruktur gefragt. Ziel der Umfrage ist es, Feedback zur Benutzerfreundlichkeit und Verständlichkeit des Plugins bzw. der Datenstruktur zu erhalten.
Die Studie wird im Rahmen der Masterarbeit „Entwicklung einer Datenstruktur zur Klassifizierung unterschiedlicher Übungsaufgaben-Elemente“ von Fabian Siebes an der RWTH Aachen durchgeführt.

**Analyse und Verwendung der während der Umfrage gesammelten Informationen**
Die Umfrage wird per Zoom oder persönlich durchgeführt. Die Teilnahme ist freiwillig. Die Teilnehmer können die Studie jederzeit beenden oder die Beantwortung von Fragen ohne Angabe von Gründen verweigern. Die Daten werden pseudonymisiert erhoben, so dass die Teilnehmer nicht identifiziert werden können. Die Teilnehmer versichern, dass sie während des Interviews keine geschützten oder geschäftskritischen Informationen preisgeben werden. Die gesammelten Informationen werden für wissenschaftliche Zwecke verwendet und können in Publikationen zitiert werden.

☐ Ich bin mit dem Vorstehenden einverstanden

# Einleitung

In der Masterarbeit wurde eine Datenstruktur zur Repräsentation von Aufgaben aus Übungsblättern entworfen.
Das Ziel der Arbeit besteht darin, die Elemente einer Aufgabe mit zusätzlichem Kontext zu klassifizieren, damit dieser Kontext für die Generierung von Feedback genutzt werden kann.
Der Fokus dieser Arbeit liegt auf Übungsblättern, die in Programmierkursen eingesetzt werden.

Diese Klassifizierung kann beispielsweise bei der Verwendung von LLMs hilfreich sein, da gefiltert werden kann, welche Teile relevant für die Feedback-Generierung sind. Um Ressourcen zu sparen, kann der Input für ein LLM verkürzt werden, indem nur die für die Generierung von Feedback notwendigen Informationen bereitgestellt werden.

# Erläuterung der Datenstruktur

Die Datenstruktur ermöglicht es, in einem Übungsblatt Aufgaben und Teilaufgaben zu erfassen und innerhalb einer Aufgabe die Aufgabenelemente zu klassifizieren.

Jede Aufgabe erhält einen Namen, wie zum Beispiel 'Aufgabe 5' oder 'b)', und es können externe Dateien angehängt werden. Eine Aufgabe besteht aus einzelnen Elementen oder weiteren Teilaufgaben. Jedes Element einer Aufgabe wird einem bestimmten Typ zugeordnet und je nach Typ können noch optionale zusätzliche Informationen erfasst werden.

Es gibt folgende Typen zur Auswahl:

- **Question** - Dieser Absatz beschreibt die Frage oder die Aufforderung an die Studierenden.
- **Constraint** - Dies ist ein Zusatz zu der gegebenen Aufgabe und gibt oft weitere Kontextinformationen oder Einschränkungen zu der Aufgabe.
- **Hint** - Diese Information soll den Studierenden bei der Lösung der Aufgabe helfen, indem sie z. B. bestimmte Funktionen angibt, die zu berücksichtigen sind.
- **Answer** - Hier können die Studierenden ihre Antwort eintragen. Teile der

Antwort können bereits vorgegeben sein.

- **Solution** - Für eine Aufgabe kann es mehrere Lösungsbeispiele geben. Wählen Sie die Lösung, die keine Alternative ist, als bevorzugte Lösung.
- **Ignore / Title / Information** - Beschreibt Aufgabentitel oder andere Informationen, die für die Bearbeitung der Aufgaben nicht relevant sind.
- **Initialization** - Dieser Abschnitt kann Codeschnipsel enthalten, die die Umgebung für die Aufgabe initialisieren und nicht geändert werden müssen.
- **Explanation** - Dies sind Informationen, die neue Konzepte erklären oder z.B. neue Formeln liefern, die zur Lösung der Aufgabe benötigt werden.
- **Example** - Beispiele sind oft Code, der der erwarteten Lösung ähnlich ist und den Studierenden bei der Formulierung ihrer eigenen Lösung helfen kann.
- **Selftests** - Dies sollte den Studierenden helfen, ihre Lösung selbst zu überprüfen. Dies kann ein Testfall sein oder auch nur eine Beschreibung der erwarteten Ausgabe.
- **Grader** - Bewertungszellen können eine Punktzahl für Studierende berechnen und werden den Studierenden oft nicht zur Verfügung gestellt.

**1. Plugin in JupyterLab testen**

Um die Klassifizierung von Übungsblättern, die als Jupyter Notebooks vorliegen, zu erleichtern, wurde ein Plugin für JupyterLab entwickelt. Das Plugin unterstützt die Erfassung und den Export der Datenstruktur für ein Übungsblatt in Form eines Jupyter Notebooks.

Bitte verwenden Sie das Plugin, um das vorgegebene Notebook zu klassifizieren. Das Plugin befindet sich in der **rechten Seitenleiste unter ‚Classify Task Elements'**. Verwenden Sie dabei möglichst die aufgelisteten Features, damit Sie später ein Feedback zu den einzelnen Funktionen geben können.

Wenn gewünscht, können Anmerkungen zu den einzelnen Punkten gemacht werden. Nach dem Testen des Plugins wird es jedoch auch einige Fragen zur Benutzerfreundlichkeit geben.

- [ ] 1. Generieren Sie die Aufgabenstruktur aus dem Inhaltsverzeichnis.

ggf. Anmerkungen

- [ ] 2. Wie Sie sehen können, wurde der Titel des Übungsblattes fälschlicherweise als Aufgabe interpretiert. „Introduction to Programming for Data-driven Sciences" sollte jedoch keine eigenständige Aufgabe sein. Diese Aufgabe sollte daher entfernt werden.

ggf. Anmerkungen

- [ ] 3. Entfernen Sie ebenfalls die Angabe der erreichbaren Punkte aus dem Titel der Aufgabe in der Struktur.

ggf. Anmerkungen

- [ ] 4. Für die Aufgabe 1 a) wird die Datei „read_and_clean.txt" benötigt und für die Aufgabe 1 b) die Datei „convert_to_csv.pickle". Bitte verknüpfen Sie die vorhandenen Dateien mit der jeweiligen Aufgabe.

ggf. Anmerkungen

- [ ] 5. Geben Sie bei Aufgabe 2c an, dass sie auf dem Ergebnis von 2a und b aufbaut.

ggf. Anmerkungen

- [ ] 6. Klassifizieren Sie nun die Aufgabenelemente bzw. Zellen aus Aufgabe 2 b) und füllen Sie gegebenenfalls die weiteren Attribute zur jeweiligen Klassifizierung aus.

ggf. Anmerkungen

**2. System Usability Score**

| | Stimme überhaupt nicht zu | | | | Stimme voll und ganz zu |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Ich denke, dass ich dieses Produkt häufig verwenden möchte | ○ | ○ | ○ | ○ | ○ |
| Ich fand das Produkt unnötig komplex | ○ | ○ | ○ | ○ | ○ |
| Ich dachte, das Produkt war einfach zu bedienen | ○ | ○ | ○ | ○ | ○ |
| Ich denke, dass ich die Unterstützung einer technischen Person brauche, um dieses Produkt nutzen zu können | ○ | ○ | ○ | ○ | ○ |
| Ich fand, die verschiedenen Funktionen in diesem Produkt waren gut integriert | ○ | ○ | ○ | ○ | ○ |
| Ich dachte, dass dieses Produkt nicht konsistent genug war | ○ | ○ | ○ | ○ | ○ |
| Ich würde mir vorstellen, dass die meisten Leute sehr schnell lernen würden, dieses Produkt zu benutzen | ○ | ○ | ○ | ○ | ○ |
| Ich fand dieses Produkt sehr umständlich zu benutzen | ○ | ○ | ○ | ○ | ○ |
| Ich habe mich sehr selbstsicher gefühlt, dieses Produkt zu verwenden | ○ | ○ | ○ | ○ | ○ |
| Ich musste eine Menge Dinge lernen, bevor ich mit diesem Produkt loslegen konnte | ○ | ○ | ○ | ○ | ○ |

### 3. Detaillierte Fragen zur Nutzung

Die automatische Erfassung der Struktur ist hilfreich

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Das Anpassen der Aufgabenstruktur ist intuitiv gestaltet

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Es war direkt ersichtlich, welche Aufgabenelemente/Zellen zu einer Aufgabe in der Struktur gehören

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Ich habe schnell herausgefunden, wie ich den Namen einer Aufgabe in der Struktur ändern kann

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Es war für mich einfach, die passenden Dateien der jeweiligen Aufgabe anzuhängen

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Ich habe verstanden, was eine Abhängigkeit zwischen einer Aufgabe und der vorhergehenden Aufgabe bedeutet

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Ich habe verstanden, wie ich die Abhängigkeit einer Aufgabe von der vorherigen Aufgabe angeben kann

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Das Zuweisen eines Typs zu einer Zelle war für mich intuitiv

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

Wenn vorhanden, habe ich die weiteren Felder für diesen Typ ausgefüllt

| Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu | *Nicht getestet* |
|---|---|---|---|---|---|

**4. Es gab einige Features, die für mich nicht verständlich waren und weitere Erklärungen benötigt hätten. Oder ich habe weitere Anmerkungen zu den Features.**

## 5. Aufteilung der Typen der Aufgabenelemente

| | | | | | |
|---|---|---|---|---|---|
| Die Unterteilung in die vorgegebenen Typen war zu detailliert | Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu |
| Ich denke, dass einige Aufgaben nicht in die Typen eingeteilt werden können | Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu |
| Ich habe mir oft eine feinere Unterteilung der Typen gewünscht | Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu |
| Die Zuordnung der Typen zu den Aufgabenelementen war einfach | Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu |
| Es gab Überschneidungen zwischen den Typen | Stimme überhaupt nicht zu | Stimme nicht zu | Stimme weder zu noch lehne ich ab | Stimme zu | Stimme voll und ganz zu |

**6. Sonstiges Anmerkungen zur Aufteilung in die Aufgabenklassen**

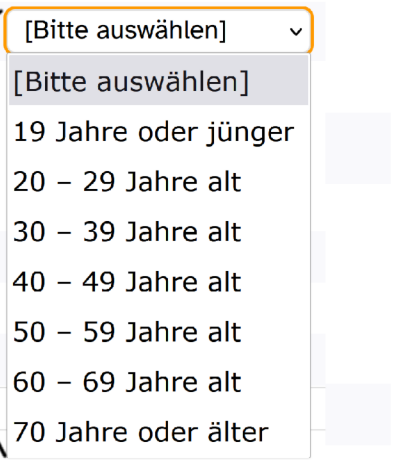
Z.B. eine Liste fehlender oder redundanter Aufgabenklassen

**7. Ich denke, dass die Klassifizierung der einzelnen Aufgabenbereiche auch in anderen Bereichen hilfreich sein könnte.**

Folgende Bereiche würden mir dazu einfallen:

**8. Alter**

Ich bin [Bitte auswählen]




**9. Ich bin/war ...**

- [ ] Dozent
- [ ] Tutor
- [ ] Student
- [ ] Anderes

**10. Bezug zu Übungsblättern**

- [ ] Ich bearbeite regelmäßig Übungsblätter.
- [ ] Ich habe bereits Übungsblätter erstellt.
- [ ] Ich habe bereits Übungsblätter korrigiert.
- [ ] Ich nutze die Ergebnisse von den korrigierten Übungsblättern

  Z.B. zur Verbesserung der Vorlesung

- [ ] Keine der oben genannten Optionen trifft auf mich zu.

**11. Wie oft arbeiten Sie mit Jupyter Notebooks?**

- ○ Ich habe bisher keine Erfahrung mit JupyterLab oder Jupyter Notebooks
- ○ Ich habe schon mit Jupyter Notebooks gearbeitet, aber noch nicht in einer JupyterLab Umgebung
- ○ Es ist schon eine Weile her, dass ich in JupyterLab gearbeitet habe
- ○ Ich arbeite gelegentlich in JupyterLab
- ○ Ich arbeite wöchentlich in JupyterLab
- ○ Ich arbeite täglich in JupyterLab

**Letzte Seite**

# Vielen Dank für Ihre Teilnahme!

Wir möchten uns ganz herzlich für Ihre Mithilfe bedanken.

Ihre Antworten wurden gespeichert, Sie können das Browser-Fenster nun schließen.

Fabian Siebes, RWTH Aachen – 2024

# Anhang D Evaluationsfragebogen (en)



**Statement of Consent**

**Purpose and content of the survey**
The survey asks participants for their opinion on the JupyterLab plugin “Classify Task Elements” and the underlying data structure. The aim of the survey is to get feedback on the usability and understandability of the plugin and the data structure. The study is being conducted as part of the master’s thesis “Development of a Data Structure for the Classification of Different Exercise Task Elements” by Fabian Siebes at RWTH Aachen University.

**Analysis and use of the information collected during the survey**
The survey will be conducted via Zoom or in person. Participation is voluntary. Participants can end the study at any time or refuse to answer questions without giving reasons. Data will be collected using pseudonyms so that participants cannot be identified. Participants agree not to disclose any proprietary or commercially sensitive information during the interview. The information collected will be used for scientific purposes and may be cited in publications.

☐ I agree to the above

# Introduction

In the master's thesis, a data structure was designed to represent tasks from exercise sheets.
The aim of the thesis is to classify the elements of a task with additional context so that this context can be used to generate feedback.
The focus of this work is on exercise sheets that are used in programming courses.

This classification can be helpful when using LLMs, for example, as it is possible to filter which parts are relevant for feedback generation. To save resources, the input for an LLM can be shortened by providing only the information necessary for generating feedback.

# Explanation of the data structure

The data structure makes it possible to record tasks and subtasks in an exercise sheet and to classify the task elements within a task.

Each task is given a name, such as 'Task 5' or 'b)', and external files can be attached. A task consists of individual elements or further subtasks. Each element of a task is assigned to a specific type and, depending on the type, additional optional information can be entered.

The following types are available:

- **Question** - This paragraph describes the question or the prompt for the student.
- **Constraint** - This is an addition of the given task and often it gives more context information or restrictions to the task.
- **Hint** - This information should help students to solve the task by providing specific functions to consider for example.
- **Answer** - Students can enter their answer here. Parts of the answer may already be given.
- **Solution** - Multiple exemplary solutions may exist for a task. Choose the solution that is not an alternative as the preferred solution.
- **Ignore / Title / Information** - Describes task titles or other information that is not relevant to completing the tasks.

- **Initialization** - This section may contain code snippets that initialize the environment for the task and do not need to be modified.
- **Explanation** - These are informations that explain new concepts or provide e.g. new formulas needed to solve the task.
- **Example** - Examples are often code that is similar to the expected solution and can assist students in formulating their own solution.
- **Selftests** - This should help the students to validate their solution themselves. This can be a test case or just a description of the expected output.
- **Grader** - Grading cells can calculate a score for a student and are often not provided to the students.

**1. Test plugin in JupyterLab**

A plugin for JupyterLab has been developed to facilitate the classification of exercise sheets that are available as Jupyter notebooks. The plugin supports the capture and export of the data structure for an exercise sheet in the form of a Jupyter notebook.

Please use the plugin to classify the given notebook. The plugin is located in the **right sidebar under 'Classify Task Elements'**. If possible, use the features listed below so that you can provide feedback on the individual functions later.

If desired, comments can be made on the individual points. After testing the plugin, however, there will also be some questions about user-friendliness.

- [ ] 1. Generate the task structure from the table of contents.
  Comments if applicable
- [ ] 2. As you can see, the title of the exercise sheet was incorrectly interpreted as a task. However, "Introduction to Programming for Data-driven Sciences" should not be an independent task. This task should therefore be removed.
  Comments if applicable
- [ ] 3. Also remove the indication of the achievable points from the title of the task in the structure.
  Comments if applicable
- [ ] 4. The file "read_and_clean.txt" is required for task 1 a) and the file "convert_to_csv.pickle" for task 1 b). Please link the existing files to the respective task.
  Comments if applicable
- [ ] 5. Indicate for task 2c that it is based on the result of 2a and b.
  Comments if applicable
- [ ] 6. Now classify the task elements resp. cells from task 2 b) and, if necessary, fill in the other attributes for the respective classification.
  Comments if applicable

**2. System Usability Score**

| | strongly disagree 1 | 2 | 3 | 4 | strongly agree 5 |
|---|---|---|---|---|---|
| I think that I would like to use this system frequently | ○ | ○ | ○ | ○ | ○ |
| I found the system unnecessarily complex | ○ | ○ | ○ | ○ | ○ |
| I thought the system was easy to use | ○ | ○ | ○ | ○ | ○ |
| I think that I would need the support of a technical person to be able to use this system | ○ | ○ | ○ | ○ | ○ |
| I found the various functions in this system were well integrated | ○ | ○ | ○ | ○ | ○ |
| I thought there was too much inconsistency in this system | ○ | ○ | ○ | ○ | ○ |
| I would imagine that most people would learn to use this system very quickly | ○ | ○ | ○ | ○ | ○ |
| I found the system very cumbersome to use | ○ | ○ | ○ | ○ | ○ |
| I felt very confident using the system | ○ | ○ | ○ | ○ | ○ |
| I needed to learn a lot of things before I could get going with this system | ○ | ○ | ○ | ○ | ○ |

### 3. Detailed questions on usage

Automatic detection of the structure is helpful

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

Customising the task structure is intuitive

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

It was immediately clear which task elements/cells belong to a task in the structure

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

I quickly found out how to change the name of a task in the structure

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

It was easy for me to attach the appropriate files to the respective task

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

I understood what a dependency between a task and the previous task means

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

I understood how to specify the dependency of a task on the previous task

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

Assigning a type to a cell was intuitive for me

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

If available, I filled in the other fields for that type

| Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree | *Not tested* |
|---|---|---|---|---|---|

**4. There were some features that were not understandable to me and would have required further explanation. Or I have further comments on the features.**

## 5. Division of the types of task elements

| | | | | | |
|---|---|---|---|---|---|
| The subdivision into the given types was too detailed | Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree |
| I think that some tasks cannot be classified into the types | Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree |
| I often wished for a finer subdivision of the types | Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree |
| The assignment of the types to the task elements was simple | Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree |
| There were overlaps between the types | Strongly disagree | Do not agree | I neither agree nor disagree | Agree | Strongly agree |

## 6. Other Notes on the division into task classes

E.g. a list of missing or redundant task classes

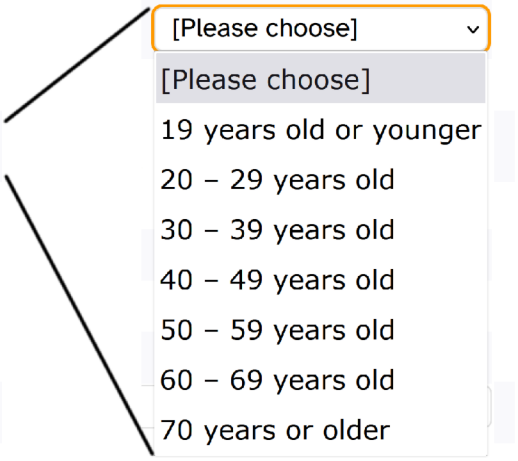
**7. I think that the categorisation of the individual task areas could also be helpful in other areas.**

The following areas come to mind:

**Page 06**

**8. Age**

I am [Please choose]




**9. I am/was a ...**

- [ ] Dozent
- [ ] Tutor
- [ ] Student
- [ ] Other

**10. Reference to exercise sheets**

- [ ] I regularly work on exercise sheets.
- [ ] I have already created exercise sheets.
- [ ] I have already corrected exercise sheets.
- [ ] I use the results from the corrected exercise sheets

  E.g. to improve the lecture

- [ ] None of the options listed above apply to me.

**11. How often do you work with Jupyter notebooks?**

- ○ I have no experience with JupyterLab or Jupyter Notebooks so far
- ○ I have already worked with Jupyter Notebooks, but not yet in a JupyterLab environment
- ○ It has been a while since I have worked in JupyterLab
- ○ I work in JupyterLab occasionally
- ○ I work in JupyterLab weekly
- ○ I work in JupyterLab daily

**Last Page**

# Thank you for your participation!

We would like to thank you very much for your help.

Your answers have been saved, you can now close the browser window.

Fabian Siebes, RWTH Aachen – 2024

# Anhang E XML-Schema der Datenstruktur

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

  <xs:simpleType name="DependencyType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="description"></xs:enumeration>
      <xs:enumeration value="result"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="QStyle">
    <xs:restriction base="xs:string">
      <xs:enumeration value="multiple-choice"></xs:enumeration>
      <xs:enumeration value="write-code"></xs:enumeration>
      <xs:enumeration value="explain-code"></xs:enumeration>
      <xs:enumeration value="draw"></xs:enumeration>
      <xs:enumeration value="parsons-problem"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="HintType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="constraint"></xs:enumeration>
      <xs:enumeration value="explanation"></xs:enumeration>
      <xs:enumeration value="example"></xs:enumeration>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Dependencies">
    <xs:sequence>
      <xs:element name="dependency" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="idref" type="xs:long" use="required" />
          <xs:attribute name="type" type="DependencyType" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ExternalFiles">
    <xs:sequence>
      <xs:element name="file" minOccurs="0" maxOccurs="unbounded">
```

```
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="mimetype" type="xs:string" use="required" />
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Structure">
  <xs:sequence>
    <xs:element name="question" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="qstyle" type="QStyle" use="optional">
            </xs:attribute>
            <xs:attribute name="qclosed" type="xs:boolean" use="optional" />
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

    <xs:element name="hint" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="type" type="HintType" use="optional" />
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

    <xs:element name="answer" minOccurs="0" maxOccurs="unbounded">
      <xs:simpleType>
        <xs:restriction base="xs:string">
        </xs:restriction>
      </xs:simpleType>
    </xs:element>

    <xs:element name="solution" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="preferredSolution" type="xs:boolean" use="
                ↪ optional" />
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
```

```
96
97      <xs:element name="info" minOccurs="0" maxOccurs="unbounded">
98        <xs:simpleType>
99          <xs:restriction base="xs:string">
100         </xs:restriction>
101       </xs:simpleType>
102     </xs:element>
103
104     <xs:element name="init" minOccurs="0" maxOccurs="unbounded">
105       <xs:simpleType>
106         <xs:restriction base="xs:string">
107         </xs:restriction>
108       </xs:simpleType>
109     </xs:element>
110
111     <xs:element name="selftests" minOccurs="0" maxOccurs="unbounded">
112       <xs:simpleType>
113         <xs:restriction base="xs:string">
114         </xs:restriction>
115       </xs:simpleType>
116     </xs:element>
117
118     <xs:element name="grader" minOccurs="0" maxOccurs="unbounded">
119       <xs:complexType>
120         <xs:simpleContent>
121           <xs:extension base="xs:string">
122             <xs:attribute name="points" type="xs:decimal" use="optional" />
123             <xs:attribute name="maxPoints" type="xs:decimal" use="optional" />
124           </xs:extension>
125         </xs:simpleContent>
126       </xs:complexType>
127     </xs:element>
128
129     <xs:element name="task" type="Task" minOccurs="0" maxOccurs="unbounded"></
        ↪ xs:element>
130   </xs:sequence>
131 </xs:complexType>
132
133 <xs:complexType name="Task">
134   <xs:all>
135     <xs:element name="dependencies" type="Dependencies" />
136     <xs:element name="externalFiles" type="ExternalFiles" />
137     <xs:element name="structure" type="Structure">
138     </xs:element>
139   </xs:all>
140   <xs:attribute name="id" type="xs:long" use="required" />
141   <xs:attribute name="bullet" type="xs:string" use="required" />
142   <xs:attribute name="maxPoints" type="xs:decimal" use="optional" />
143   <xs:attribute name="points" type="xs:decimal" use="optional" />
144 </xs:complexType>
145
146 <xs:element name="task" type="Task" />
147 </xs:schema>
```

# Anhang F Literatur


[1] Rishabh Balse et al. „Investigating the Potential of GPT-3 in Providing Feedback for Programming Assessments“. In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2023. New York, NY, USA: Association for Computing Machinery, Juni 2023, S. 292–298. ISBN: 9798400701382. DOI: 10.1145/3587102.3588852. (Besucht am 11. 11. 2023).

[2] Aaron Bangor, Philip Kortum und James Miller. „Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale“. In: *Journal of Usability Studies* 4.3 (Mai 2009), S. 114–123.

[3] Matt Bower. „A Taxonomy of Task Types in Computing“. In: *ACM SIGCSE Bulletin* 40.3 (Juni 2008), S. 281–285. ISSN: 0097-8418. DOI: 10.1145/1597849.1384346. (Besucht am 10. 11. 2023).

[4] John Brooke. „SUS: A Quick and Dirty Usability Scale“. In: *Usability Eval. Ind.* 189 (Nov. 1995).

[5] James P. Byrnes. *Minds, Brains, and Learning: Understanding the Psychological and Educational Relevance of Neuroscientific Research*. Guilford Press, Apr. 2001. ISBN: 978-1-57230-652-3.

[6] Alberto Cardoso, Joaquim Leitão und César Teixeira. „Using the Jupyter Notebook as a Tool to Support the Teaching and Learning Processes in Engineering Courses“. In: *The Challenges of the Digital Transformation in Education*. Hrsg. von Michael E. Auer und Thrasyvoulos Tsiatsos. Cham: Springer International Publishing, 2019, S. 227–236. ISBN: 978-3-030-11935-5. DOI: 10.1007/978-3-030-11935-5_22.

[7] Paul Denny, Andrew Luxton-Reilly und Beth Simon. „Evaluating a New Exam Question: Parsons Problems“. In: *Proceedings of the Fourth International Workshop on Computing Education Research*. ICER ’08. New York, NY, USA: Association for Computing Machinery, Sep. 2008, S. 113–124. ISBN: 978-1-60558-216-0. DOI: 10.1145/1404520.1404532. (Besucht am 15. 03. 2024).

[8] Norm Friesen. „Interoperability and Learning Objects: An Overview of E-Learning Standardization“. In: *Interdisciplinary Journal of E-Learning and Learning Objects* 1.1 (Jan. 2005), S. 23–31. ISSN: 1552-2237. URL: https://www.learntechlib.org/p/44864/ (besucht am 16. 04. 2024).

[9] Meiyuzi Gao, Phil Kortum und Frederick Oswald. „Multi-Language Toolkit for the System Usability Scale“. In: *International Journal of Human-Computer Interaction* 36 (Aug. 2020), S. 1–19. DOI: 10.1080/10447318.2020.1801173.

[10] Richard Gluga et al. *Coming to Terms with Bloom: An Online Tutorial for Teachers of Programming Fundamentals*. 2012.

[11] *Google Trends Large Language Model*. URL: https://trends.google.com/trends/explore?date=today%205-y&geo=DE&q=%2Fg%2F11kc9956b3&hl=de (besucht am 25. 04. 2024).

[12] Cécile Hardebolle. *Exercise Worksheets with Jupyter Notebooks*. URL: https://www.epfl.ch/education/educational-initiatives/jupyter-notebooks-for-education/teaching-and-learning-with-jupyter-notebooks/exercise-worksheets-with-jupyter-notebooks/ (besucht am 25. 04. 2024).

[13] James Harland. *A Database for Storing and Retrieving Classified and Benchmarked Exam Questions for ICT Courses*. 2014.

[14] John Hattie und Helen Timperley. „The Power of Feedback“. In: *Review of Educational Research* 77.1 (März 2007), S. 81–112. ISSN: 0034-6543. DOI: 10.3102/003465430298487. (Besucht am 06. 03. 2024).

[15] Orit Hazzan, Tami Lapidot und Noa Ragonis. „Types of Questions in Computer Science Education“. In: *Guide to Teaching Computer Science*. London: Springer London, 2011, S. 143–163. ISBN: 978-0-85729-442-5 978-0-85729-443-2. URL: https://link.springer.com/10.1007/978-0-85729-443-2_9 (besucht am 10. 11. 2023).

[16] Orit Hazzan, Noa Ragonis und Tami Lapidot. *Guide to Teaching Computer Science: An Activity-Based Approach*. Cham: Springer International Publishing, 2011. ISBN: 978-3-030-39359-5 978-3-030-39360-1. URL: http://link.springer.com/10.1007/978-3-030-39360-1 (besucht am 28. 11. 2023).

[17] Arto Hellas et al. „Exploring the Responses of Large Language Models to Beginner Programmers’ Help Requests“. In: *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*. Bd. 1. ICER ’23. New York, NY, USA: Association for Computing Machinery, Sep. 2023, S. 93–105. ISBN: 978-1-4503-9976-0. DOI: 10.1145/3568813.3600139. (Besucht am 11. 11. 2023).

[18] *1EdTech Learning Design Best Practice and Implementation Guide Version 1.0 Final | IMS Global Learning Consortium*. URL: https://www.imsglobal.org/learningdesign/ldv1p0/imsld_bestv1p0.html#1503660 (besucht am 19. 04. 2024).

[19] Johan Jeuring et al. „Towards Giving Timely Formative Feedback and Hints to Novice Programmers“. In: *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*. ITiCSE-WGR ’22. New York, NY, USA: Association for Computing Machinery, Dez. 2022, S. 95–115. ISBN: 9798400700101. DOI: 10.1145/3571785.3574124. (Besucht am 17. 11. 2023).

[20] *JSON Schema: A Media Type for Describing JSON Documents*. URL: https://json-schema.org/draft/2020-12/json-schema-core#rfc.section.7.7.1.1 (besucht am 11. 12. 2023).

[21] *Project Jupyter Documentation — Jupyter Documentation 4.1.1 Alpha Documentation*. URL: https://docs.jupyter.org/en/latest/ (besucht am 15. 03. 2024).

[22] *Jupyter Nbviewer*. URL: https://nbviewer.org/ (besucht am 15. 03. 2024).

[23] *Jupyter Notebook Documentation — Jupyter Notebook 7.1.2 Documentation*. URL: https://jupyter-notebook.readthedocs.io/en/latest/ (besucht am 15. 03. 2024).

[24] *Working with Jupyter Notebooks in Visual Studio Code*. URL: https://code.visualstudio.com/docs/datascience/jupyter-notebooks (besucht am 15. 03. 2024).

[25] *Project Jupyter*. URL: https://jupyter.org (besucht am 19. 10. 2023).

[26] *Project Jupyter | About*. URL: https://jupyter.org/about (besucht am 14. 03. 2024).

[27] *JupyterFrontEndPlugin | @jupyterlab*. URL: https://jupyterlab.readthedocs.io/en/stable/api/types/application.JupyterFrontEndPlugin.html (besucht am 01. 04. 2024).

[28] *Project Jupyter | JupyterHub*. URL: https://jupyter.org/hub (besucht am 15. 03. 2024).

[29] *Common Extension Points — JupyterLab 4.1.5 Documentation*. URL: https://jupyterlab.readthedocs.io/en/stable/extension/extension_points.html#core-plugins (besucht am 01. 04. 2024).

[30] *Develop Extensions — JupyterLab 4.0.7 Documentation*. URL: https://jupyterlab.readthedocs.io/en/stable/extension/extension_dev.html#developer-extensions (besucht am 19. 10. 2023).

[31] *JupyterLab Documentation — JupyterLab 4.2.0a0 Documentation*. URL: https://jupyterlab.readthedocs.io/en/latest/ (besucht am 15. 03. 2024).

[32] *Jupyterlab/Extension-Template*. JupyterLab. März 2024. URL: https://github.com/jupyterlab/extension-template (besucht am 31. 03. 2024).

[33] *React — JupyterLab 4.2.0a2 Documentation*. URL: https://jupyterlab.readthedocs.io/en/latest/extension/virtualdom.html (besucht am 31. 03. 2024).

[34] *Table Of Contents — JupyterLab 3.6.7 Documentation*. URL: https://jupyterlab.readthedocs.io/en/3.6.x/user/toc.html (besucht am 30. 03. 2024).

[35] Florian Kapp und Carsten Schulte. „Einsatz von Jupyter Notebooks am Beispiel eines fiktiven Kriminalfalls“. In: *Informatik für alle*. Gesellschaft für Informatik, 2019, S. 247–256. ISBN: 978-3-88579-682-4. URL: https://dl.gi.de/handle/20.500.12116/28931 (besucht am 23. 03. 2024).

[36] Hieke Keuning, Johan Jeuring und Bastiaan Heeren. „A Systematic Literature Review of Automated Feedback Generation for Programming Exercises“. In: *ACM Transactions on Computing Education* 19.1 (Sep. 2018), 3:1–3:43. DOI: 10.1145/3231711. (Besucht am 11. 11. 2023).

[37] Thomas Kluyver et al. *Jupyter Notebooks—a Publishing Format for Reproducible Computational Workflows*. 2016.

[38] Franz Knülle. *Answer to "Notebook Validation Failed: Additional Properties Are Not Allowed ('id' Was Unexpected):"* Feb. 2022. URL: https://stackoverflow.com/a/71217353/21597449 (besucht am 11. 12. 2023).

[39] Rob Koper. „Modeling Units of Study from a Pedagogical Perspective: The Pedagogical Meta-Model behind EML". In: *Modeling units of study from a pedagogical perspective: the pedagogical meta-model behind EML* (Nov. 2001), S. 1–40.

[40] Nguyen-Thinh Le und Niels Pinkwart. *Towards-a-Classification-for-Programming-Exercises.Pdf*. URL: https://publications.informatik.hu-berlin.de/archive/cses/publications/Towards-a-Classification-for-Programming-Exercises.pdf (besucht am 30. 01. 2024).

[41] *Draft Standard for Learning Object Metadata*. Juli 2002. URL: https://web.archive.org/web/20130701001100/http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf (besucht am 19. 04. 2024).

[42] Mark Liffiton et al. *CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes*. 2023. DOI: 10.48550/ARXIV.2308.06921.

[43] *IPlugin | @lumino - V2024.3.25*. URL: https://lumino.readthedocs.io/en/latest/api/interfaces/application.IPlugin.html (besucht am 01. 04. 2024).

[44] *ISignal | @lumino - V2024.3.25*. URL: https://lumino.readthedocs.io/en/latest/api/interfaces/signaling.ISignal.html (besucht am 01. 04. 2024).

[45] Andrew Luxton-Reilly et al. „Introductory Programming: A Systematic Literature Review". In: *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018 Companion. New York, NY, USA: Association for Computing Machinery, Juli 2018, S. 55–106. ISBN: 978-1-4503-6223-8. DOI: 10.1145/3293881.3295779. (Besucht am 30. 11. 2023).

[46] *Markdown.de | Markdown Syntax-Dokumentation*. URL: https://markdown.de/ (besucht am 20. 04. 2024).

[47] Tilman Michaeli und Ralf Romeike. „Improving Debugging Skills in the Classroom: The Effects of Teaching a Systematic Debugging Process". In: *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*. WiPSCE '19. New York, NY, USA: Association for Computing Machinery, Okt. 2019, S. 1–7. ISBN: 978-1-4503-7704-1. DOI: 10.1145/3361721.3361724. (Besucht am 15. 04. 2024).

[48] Susanne Narciss. *Feedback Strategies for Interactive Learning Tasks*. Jan. 2008.

[49] *The Notebook File Format — Nbformat 5.9 Documentation*. URL: https://nbformat.readthedocs.io/en/latest/format_description.html (besucht am 17. 10. 2023).

[50] *Cell ID Addition to Notebook Format — Jupyter Enhancement Proposals*. URL: https://jupyter.org/enhancement-proposals/62-cell-id/cell-id.html (besucht am 11. 12. 2023).

[51] Xavier Ochoa und Stefaan Ternier. „Technical Learning Infrastructure, Interoperability and Standards“. In: *Technology Enhanced Learning: Research Themes*. Hrsg. von Erik Duval, Mike Sharples und Rosamund Sutherland. Cham: Springer International Publishing, 2017, S. 145–155. ISBN: 978-3-319-02600-8. DOI: 10.1007/978-3-319-02600-8_14. (Besucht am 16. 04. 2024).

[52] *OpenAI Pricing*. URL: https://openai.com/pricing (besucht am 20. 04. 2024).

[53] Maciej Pankiewicz und Ryan S. Baker. *Large Language Models (GPT) for Automating Feedback on Programming Assignments*. 2023. DOI: 10.48550/ARXIV.2307.00150.

[54] Andrew Petersen, Michelle Craig und Daniel Zingaro. „Reviewing CS1 Exam Question Content“. In: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’11. New York, NY, USA: Association for Computing Machinery, März 2011, S. 631–636. ISBN: 978-1-4503-0500-6. DOI: 10.1145/1953163.1953340. (Besucht am 29. 11. 2023).

[55] *Jupyter Notebook Support | PyCharm*. URL: https://www.jetbrains.com/help/pycharm/ipython-notebook-support.html (besucht am 15. 03. 2024).

[56] Paul Reiser et al. *ProFormA 2.0 – ein Austauschformat für automatisiert bewertete Programmieraufgaben und für deren Einreichungen und Feedback*. 2019. DOI: 10.18420/ABP2019-6.

[57] Lianne Roest, Hieke Keuning und Johan Jeuring. *Next-Step Hint Generation for Introductory Programming Using Large Language Models*. Dez. 2023. DOI: 10.48550/arXiv.2312.10055. arXiv: 2312.10055. (Besucht am 21. 12. 2023).

[58] Reudismam Rolim et al. „Learning Syntactic Program Transformations from Examples“. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. Mai 2017, S. 404–415. DOI: 10.1109/ICSE.2017.44. (Besucht am 16. 03. 2024).

[59] Alexander Ruf, Marc Berges und Peter Hubwieser. „Classification of Programming Tasks According to Required Skills and Knowledge Representation“. In: *Informatics in Schools. Curricula, Competences, and Competitions*. Hrsg. von Andrej Brodnik und Jan Vahrenhold. Bd. 9378. Cham: Springer International Publishing, 2015, S. 57–68. ISBN: 978-3-319-25395-4 978-3-319-25396-1. URL: http://link.springer.com/10.1007/978-3-319-25396-1_6 (besucht am 08. 11. 2023).

[60] *SCORM CAM 2004*. Aug. 2009. URL: https://adlnet.gov/past-projects/scorm/#scorm-versions-and-resources.

[61] J. Sheard et al. *How difficult are exams? A framework for assessing the complexity of introductory programming exams*. Australian Computer Society (ACS), Jan. 2013. ISBN: 978-1-921770-21-0. URL: https://hdl.handle.net/10292/5151 (besucht am 10. 11. 2023).

[62] Judy Sheard et al. „Exploring Programming Assessment Instruments: A Classification Scheme for Examination Questions“. In: *Proceedings of the Seventh International Workshop on Computing Education Research*. ICER ’11. New York, NY, USA: Association for Computing Machinery, Aug. 2011, S. 33–38. ISBN: 978-1-4503-0829-8. DOI: 10.1145/2016911.2016920. (Besucht am 08. 11. 2023).

[63] Freda Shi et al. „Large Language Models Can Be Easily Distracted by Irrelevant Context“. In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Juli 2023, S. 31210–31227. URL: https://proceedings.mlr.press/v202/shi23a.html (besucht am 14. 03. 2024).

[64] Valerie J. Shute. „Focus on Formative Feedback“. In: *Review of Educational Research* 78.1 (März 2008), S. 153–189. ISSN: 0034-6543. DOI: 10.3102/0034654307313795. (Besucht am 08. 03. 2024).

[65] Rishabh Singh, Sumit Gulwani und Armando Solar-Lezama. *Automated Feedback Generation for Introductory Programming Assignments | Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2013. URL: https://dl.acm.org/doi/abs/10.1145/2491956.2462195?casa_token=a4SgCJpDDvQAAAAA:g9ImZrSX13B4P132w-HTtMlIRUl1bq1dPZ0cQTkVu7QUuCxcUS_R5Q7VN7F-bIgBq_LDm3D9VRqr (besucht am 16. 03. 2024).

[66] Sven Strickroth et al. „Wiederverwendbarkeit von Programmieraufgaben durch Interoperabilität von Programmierlernsystemen“. In: *DeLFI 2014 - Die 12. e-Learning Fachtagung Informatik*. Gesellschaft für Informatik e.V., 2014, S. 97–108. ISBN: 978-3-88579-627-5. URL: https://dl.gi.de/items/61552d79-8200-4175-b5ca-ca4efdda0f13 (besucht am 21. 03. 2024).

[67] Ryo Suzuki et al. „Exploring the Design Space of Automatically Synthesized Hints for Introductory Programming Assignments“. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’17. New York, NY, USA: Association for Computing Machinery, Mai 2017, S. 2951–2958. ISBN: 978-1-4503-4656-6. DOI: 10.1145/3027063.3053187. (Besucht am 16. 03. 2024).

[68] *Visual Studio Code: How Microsoft’s ’any OS, Any Programming Language, Any Software’ Plan Is Paying Off*. URL: https://www.zdnet.com/article/visual-studio-code-how-microsofts-any-os-any-programming-language-any-software-plan-is-paying-off/ (besucht am 15. 03. 2024).

# Anhang G  Digitaler Anhang

# Eidesstattliche Versicherung
## Declaration of Academic Integrity

Siebes, Fabian | 394747

Name, Vorname/Last Name, First Name | Matrikelnummer (freiwillige Angabe) Student ID Number (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/Masterarbeit* mit dem Titel
I hereby declare under penalty of perjury that I have completed the present paper/bachelor's thesis/master's thesis* entitled

Entwicklung einer Datenstruktur zur Klassifizierung unterschiedlicher Übungsaufgaben-Elemente

Development of a Data Structure for the Classification of Different Exercise Task Elements

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt; dies umfasst insbesondere auch Software und Dienste zur Sprach-, Text- und Medienproduktion. Ich erkläre, dass für den Fall, dass die Arbeit in unterschiedlichen Formen eingereicht wird (z.B. elektronisch, gedruckt, geplottet, auf einem Datenträger) alle eingereichten Versionen vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without unauthorized assistance from third parties (in particular academic ghostwriting. I have not used any other sources or aids than those indicated; this includes in particular software and services for language, text, and media production. In the event that the work is submitted in different formats (e.g. electronically, printed, plotted, on a data carrier), I declare that all the submitted versions are fully identical. I have not previously submitted this work, either in the same or a similar form to an examination body.

Aachen, den 26.04.2024 | Unterschrift/Signature

Ort, Datum/City, Date

*Nichtzutreffendes bitte streichen/Please delete as appropriate

**Belehrung:**
**Official Notification:**

**§ 156 StGB: Falsche Versicherung an Eides Statt**
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
**§ 156 StGB (German Criminal Code): False Unsworn Declarations**
Whosoever before a public authority competent to administer unsworn declarations (including Declarations of Academic Integrity) falsely submits such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment for a term not exceeding three years or to a fine.
**§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt**
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
**§ 161 StGB (German Criminal Code): False Unsworn Declarations Due to Negligence**
(1) If an individual commits one of the offenses listed in §§ 154 to 156 due to negligence, they are liable to imprisonment for a term not exceeding one year or to a fine.
(2) The offender shall be exempt from liability if they correct their false testimony in time. The provisions of § 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, den 26.04.2024 | Unterschrift/Signature

Ort, Datum/City, Date