Universität Leipzig
Wirtschaftswissenschaftliche Fakultät
Institut für Wirtschaftsinformatik
Prof. Dr. Ulrich Eisenecker
M.Sc. Nico Willert

Thema

Generator für Single-Choice-Aufgaben

Bachelorarbeit zur Erlangung des akademischen Grades
Bachelor of Science – Wirtschaftsinformatik

vorgelegt von: Thiemann, Jonathan

Matrikelnummer: 370 9756

Email-Adresse: jt88lalu@studserv.uni-leipzig.de

Leipzig, den 10.02.2022

Abstract

In einer Zeit von steigenden Studierendenzahlen und zunehmender Nachfrage nach individuellen Übungsmöglichkeiten, sehen sich Lehrende mit der Herausforderung konfrontiert, mit wenig Ressourcen möglichst viele Lernmöglichkeiten bereitzustellen. Computergestützte Übungen können hierfür eine ressourcenschonende Möglichkeit darstellen, und die direkte Interaktion zwischen Lehrenden und Lernenden um eine flexible und digitale Komponente ergänzen. Für eine Variation in den Single-Choice-Aufgaben kann ein Generator verwendet werden. Diese Arbeit untersucht den aktuellen Stand der Generierung von Multiple-Choice-Aufgaben und stellt einige Softwarelösungen vor.
Im Anschluss wird die Implementierung eines angepassten Generators exemplarisch erläutert anhand eines Prototyps in der Programmiersprache Python. Der Prototyp übernimmt eine Aufgabenschablone, die mehrere Antworten und Parameter enthalten kann. Daraus können unterschiedliche Aufgaben generiert werden.

Gliederung

Gliederung ..... **I**
Abbildungsverzeichnis ..... **II**
Verzeichnis der Listings ..... **II**
Tabellenverzeichnis ..... **II**
Abkürzungsverzeichnis ..... **III**
1 Einleitung **1**
1.1 Motivation 1
1.2 Zielstellung 3
1.3 Aufbau der Arbeit 4
2 Literaturüberblick **5**
2.1 Vorgehen 5
2.2 Ergebnisse 6
2.2.1 Aufgaben generieren 6
2.2.2 Prüfungen generieren 8
2.2.3 Lernobjekte erstellen und teilen 11
3 Domain Engineering **13**
3.1 Domänenanalyse 13
3.2 Domänendesign 14
3.3 Domänenimplementierung 15
4 Tool Beschreibung **16**
4.1 Programmablauf 16
4.2 Die Konfigurationsdatei 19
4.3 Besonderheiten 21
4.4 Die Log-Datei 21
4.5 Testing 24
5 Benutzung **27**
6 Fazit und Evaluation **28**
7 Ausblick **29**
Literaturverzeichnis ..... **IV**
Anhang ..... **VI**

Abbildungsverzeichnis

Abbildung 1: Bloom’sche Taxonomie 1

Abbildung 2: Beispielhafte Kreuzung 9

Abbildung 3: Beispielhafte Ausprägung 10

Abbildung 4: Merkmalmodell 14

Abbildung 5: Grober Programmablauf 16

Abbildung 6: Aufbau der Lösungsmenge mittels Kombinatorik 18

Abbildung 7: Anzahlberechnung der Lösungsmenge 24

Abbildung 8: Teilabschnitt der Lösungsmenge 26

Abbildung 9: Ordnerstruktur VIII

Verzeichnis der Listings

Listing 1: Abfragemuster für die Multiple-Choice-Frage 10

Listing 2: Beispiel einer Konfigurationsdatei 19

Listing 3: Beispielhafte Log-Datei 22

Listing 4: Prototyp ohne Dialog ausführen 28

Listing 5: Beispielhafte Konfigurationsdatei im JSON-Format IX

Listing 6: Log-Datei mit Fehlerreport X

Listing 7: Konfigurationsdatei im JSON-Format für die Überprüfung XI

Listing 8: Erwartete Log-Datei XII

Listing 9: Konfigurationsdatei im JSON-Format für den Kombinatorik-Test XIII

Listing 10: Leere Konfigurationsdatei XIV

Tabellenverzeichnis

Tabelle 1: Frage/Antwort-Templates, vereinfachte Beschreibung 11

Abkürzungsverzeichnis

GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
ILIAS	Integriertes Lern-, Informations- und Arbeitskooperations-System
JSON	JavaScript Object Notation
Moodle	Modular Object-Oriented Dynamic Learning Environment
SPL	Software Product Line
XML	Extensible Markup Language

1 Einleitung

Das Arbeiten mit dem Computer ist vermutlich eines der markantesten Merkmale des 21. Jahrhunderts. In vielen Bereichen der Gesellschaft ist die Art der Kommunikation bereits ein untrennbarer Bestandteil des alltäglichen Lebens geworden. Insbesondere in der Bildung folgt die Schüler-Lehrer-Interaktion immer stärker dieser Entwicklung. Gerade in Zeiten einer weltweiten Pandemie ist der Begriff „Homeschooling“ in aller Munde. Die allgemeinen Studierendenzahlen steigen immer weiter an, dadurch werden die Unterrichtskurse an Universitäten immer größer. Gerade weil dabei ein einheitliches Vorwissen der Studierenden nicht vorausgesetzt werden kann, wäre eine individualisierte Betreuung sinnvoll. Dies würde sicherstellen, dass eine ausreichende Interaktion und Aufrechterhaltung der Motivation gewährleistet werden (vgl. [Burr et al. 2015, 13ff]). Aufgrund des hohen Aufwandes und begrenzter Ressourcen ist eine solche Betreuung allerdings nur teilweise möglich. Wöchentliche Leistungsnachweise oder automatisierbare Übungen können dabei einen Teil der Betreuung darstellen. Diese potentielle Herangehensweise bietet den Studierenden eine regelmäßige Selbsteinschätzung und den Lehrenden eine Rückmeldung über etwaige Wissenslücken, während der notwendige Aufwand, um solche Aufgaben zu erstellen, enorm reduziert wird.

1.1 Motivation

Allgemein werden Fragen häufig nach der Bloom’schen Taxonomie eingestuft. In Abbildung 1 werden die einzelnen Bausteine Wissen, Verstehen, Anwendung, Analyse, Bewerten, Entwickeln hierarchisch dargestellt. Jeder Baustein unterscheidet sich hinsichtlich der Komplexitäts- und Spezifitätsstufen.

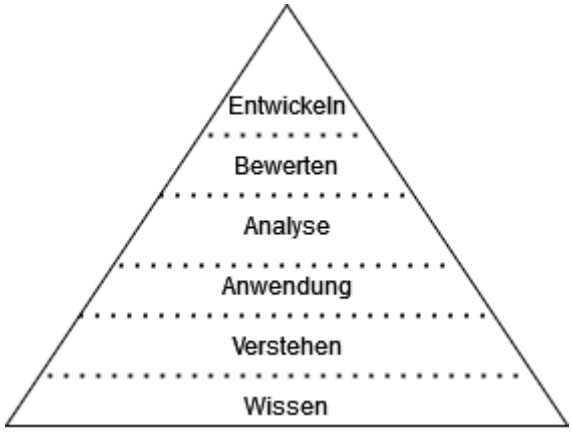

Abbildung 1: Bloom’sche Taxonomie nach [Bloom 1956]

Die einzelnen Abschnitte werden nun erläutert. Ob eine gewisse Stufe von einem Prüfling erreicht worden ist, kann mittels geeigneter Testfragen überprüft werden. Die Eigenschaften der Fragen werden in Anlehnung an [BLOOM 1956] kurz umschrieben.

Wissen: Beschreibt die Fähigkeit der Studierenden, sich bestimmte Fakten zu merken und abzurufen. - Die Testfragen konzentrieren sich auf das Erkennen und Abrufen von Informationen.

Verstehen: Bezieht sich auf die Fähigkeit der Studierenden, Kursinhalte zu bearbeiten und dabei wichtige Informationen zu extrapolieren, zu interpretieren und die Ideen anderer in eigene Worte zu fassen. - Die Testfragen konzentrieren sich auf die Replikation von Fakten, Regeln und Prinzipien

Anwendung: Die Studierenden wenden Konzepte auf andere oder erweiterte Problemstellungen an. - Die Testfragen konzentrieren sich auf die Anwendung von Fakten oder Prinzipien.

Analyse: Die Studierenden sind in der Lage, neue Informationen in ihre Einzelteile zu zerlegen und diese einzeln geordnet zu untersuchen. - Die Testfragen konzentrieren sich auf die Zerlegung eines Ganzen in seine Bestandteile.

Bewerten: Die Fähigkeit der Studierenden, die Ideen oder Grundsätze eines anderen zu betrachten und den Wert der Arbeit und den Wert der Schlussfolgerungen zu erkennen. - Die Testfragen konzentrieren sich auf die Entwicklung von Meinungen, Urteilen oder Entscheidungen.

Entwicklung: Die Studierenden sind in der Lage, verschiedene Informationen zu einem Ganzen zusammenzufügen und so ein nicht vorhandenes Muster zu schaffen war. - Die Testfragen konzentrieren sich auf die Kombination von Ideen zu einem neuen Ganzen.

Einzelnen Fragen in einer Prüfung lassen sich den eben genannten Ebenen zuordnen. In einer Prüfung sollten sich Aufgaben zu jeder Kategorie wiederfinden oder zumindest einige davon abdecken. Neben der Art der Fragen ist es aber auch entscheidend, auf welche Weise diese gestellt werden. Laut Zimmaro [2004, 19] sollte sich eine Frage auf eine Thematik fokussieren und unabhängig von anderen Aufgaben sein. Fragen sollten verständlich formuliert werden und eine erhöhte Schwierigkeit nur durch die Komplexität des behandelten Lehrstoffes erhalten. „*Be sure that the problem posed is clear and unambiguous*“ [Zimmaro 2004, 19]. „*Avoid trick or catch questions in an achievement test (Don’t waste time testing how well the student can interpret your intentions).*“ [Zimmaro 2004, 19]. Ein solches Beispiel bei Multiple-Choice-Aufgaben ist: „Keine der oben genannten“. Jede Antwortmöglichkeit sollte homogen und plausibel gehalten werden. Dabei

sollten keine Hinweise über sich gegenseitig ausschließende Antwortalternativen oder grammatikalische Hinweise durch Ergänzungen eines Satzes gegeben werden. (vgl. [Zimmaro 2004, 19])

Anhand der genannten Richtlinien und Möglichkeit der Kategorisierung für die Fragenerstellung zeigt sich, dass der Prozess bei der Entwicklung neuer Fragen sehr arbeits- und zeitintensiv sein kann. Aufgrund fehlender Ressourcen oder Zeit kann die Qualität von Fragen für den Lern- und Unterrichtsprozess beeinträchtigt werden. Generell wird angestrebt, mit einem bestimmten Aufwand ein möglichst gutes Ergebnis zu erzielen. Computergestützte Multiple-Choice-Tests sind hierbei unter Lehrenden ein weit verbreitetes Mittel zur Überprüfung von Wissen der Studierenden. Dabei können passende Fragen wiederverwendet, eine große Anzahl an Studierenden gleichzeitig erreicht und die Fragen automatisch ausgewertet werden. Um die Integrität einzelner Fragen, gesamter Prüfungen oder mehrerer Prüfungsdurchläufe zu gewährleisten, ist es notwendig, einen ausgesprochen großen Fragepool vorrätig zuhaben. Um diesen mittels begrenzter Ressourcen und endlichem Stoffumfang zu realisieren, soll durch die vorliegende Arbeit ein Generator für Single-Choice-Aufgaben konzipiert und umgesetzt werden. Dieser hat zum Vorteil, einen vergleichbaren Schwierigkeitsgrad bei sich unterscheidenden Fragen zu besitzen. Außerdem wird der Mehraufwand der Lehrenden reduziert. Ein weiterer Vorteil ist, dass wöchentliche Aufgaben eine erhöhte Motivation der Studierenden bewirken [Nagasaka 2020, 14]. Solche Aufgaben zur Selbstüberprüfung können computergesteuert an die Studierenden ausgegeben werden.

1.2 Zielstellung

Im Zuge der Arbeit soll der aktuelle Stand von Generatoren für Multiple- und Single-Choice-Aufgaben betrachtet werden. Das Ziel ist es, anhand eines Prototyps einen Generator für Single-Choice-Aufgaben zu implementieren.

Dabei sollen die im Vorhinein definierten Anforderungen analysiert und bei der Implementierung des Prototyps mit einbezogen werden.

Konkret lassen sich hierfür zwei Forschungsfragen ableiten:

F1: „*Wie ist der aktuelle Stand von Generatoren bei Multiple-Choice-Aufgaben?*“

F2: „*Wie ist ein Generator für Single-Choice-Aufgaben mit Anbindung an das ILIAS-System zu erstellen?*“

1.3 Aufbau der Arbeit

Um die Forschungsfrage 1 zu beantworten, wird zunächst ein Literaturüberblick über den aktuellen Stand von Multiple-Choice-Aufgaben im zweiten Kapitel gegeben. Dieses legt die Grundlage und Vorarbeit für die Forschungsfrage 2, auf deren Beantwortung der Fokus dieser Arbeit liegt.

Hierfür wird in den Kapiteln vier und fünf der Programmablauf des Prototyps und die Benutzung dessen erläutert.

Schlussendlich wird ein Fazit gezogen und ein Ausblick auf weiterführende Arbeitsschritte gegeben.

Im Anhang sind die Anforderungen an den Prototypen, eine Abbildung und Listings zu finden.

2 Literaturüberblick

Im folgenden Kapitel wird der aktuelle Stand von Multiple-Choice-Generatoren dargestellt. Die Erarbeitung der ausgewählten Literatur umfasst vier Schritte: Planung, Selektion, Extraktion und Ausführung (in Anlehnung an [Okoli/Schabram 2010]). In dieser Arbeit wird keine umfassende strukturierte Literaturrecherche durchgeführt. Es soll lediglich ein Überblick mittels ausgewählter Softwarebeispiele gegeben werden.

2.1 Vorgehen

Planung

Der Literaturüberblick hat die Sammlung vorhandener Erkenntnisse zum Thema Multiple-Choice-Aufgaben zum Gegenstand. Der Fokus liegt hierbei auf bereits implementierter Software oder erarbeiteten Prototypen, um Ansätze und Vorwissen für Forschungsfrage 2 übernehmen zu können.

Ziel ist somit eine Überprüfung der bestehenden Forschung, welche sich mit Generatoren für Multiple-Choice-Aufgaben beschäftigt. Aus diesem Grund wurde in Online-Literaturdatenbanken nach dem Schlüsselwort „multiple choice generator“ gesucht.

Der Literaturüberblick umfasst deutsch- und englischsprachige Literatur. Die ausgesuchten Publikationen sollten nicht mehr als fünfzehn Jahre zurückliegen, um einen aktuellen Bezug zu haben. Die Quellensuche verwendet die Online-Literaturdatenbanken IEEE Explorer, Google Scholar sowie die Universitätsbibliothek Leipzig.

Selektion

Zu Beginn des zweiten Schritts wird nach geeigneter Fachliteratur und Forschungsarbeiten gesucht mittels den Online-Literaturdatenbanken, die in der Planungsphase benannt wurden. Unter der Beachtung der in der Planung festgelegten Parameter, werden hierbei Suchanfragen über einzelne Themen und deren Teilmengen gestellt.

Weiterhin erfolgt eine Einordnung der Artikel nach ihrem Inhalt. Die wesentlichen Kriterien sind hierbei der Titel und das Abstract. Darauf werden die Artikel, die zur Beantwortung der Untersuchung dienlich sind, mit in die Literatursammlung aufgenommen. Bei der Auswahl der Artikel wird nicht darauf geachtet, ob der Anwendungsfall exakt gleich der zugrundeliegenden Arbeit ist. Dadurch können auch Abweichungen miterfasst werden. Der verwendete Suchbegriff lautet: „multiple choice generator“. Es wird bewusst „multiple choice generator“ als Suchbegriff verwendet und nicht, wie es der Titel der Arbeit vermuten lässt, Single-Choice-Generator, da in einiger Literatur, insbesondere in englischer, die

Begriffe synonym verwendet werden. Auch ist der Kern der beiden Anwendungsfälle nicht wesentlich unterschiedlich.

Extraktion

Bei den 16 nach Titel und Zusammenfassung ausgesuchten Artikeln sind nicht alle für die Beantwortung der Forschungsfrage 1 relevant. Daher werden diese zunächst eingehend auf den Inhalt und Bezug zur vorliegenden Arbeit geprüft. Im Zuge dessen wurden vier Artikel aussortiert. Im nächsten Schritt werden die Daten extrahiert und systematisch aufgeschlüsselt. Die finale Auswahl an relevanter Literatur wird anschließend in drei Themengebiete eingeteilt und dargestellt: Aufgabengenerierung, Prüfungsgenerierung, sowie Erstellung und Verbreitung von Lernobjekten.

Ausführung

Im abschließenden Schritt werden die Informationen zusammengeführt und thematisch dargestellt. Durch die vorhergehenden Schritte stehen alle nötigen Daten zur Verfügung, um die Forschungsfrage 1 beantworten zu können, die einzelnen Studien zu interpretieren und Bezüge untereinander herzustellen.

2.2 Ergebnisse

In diesem Abschnitt werden die gewonnenen Erkenntnisse dargestellt. Die geeignete Literatur wird in die oben genannten drei Themenbereiche kategorisiert. Die aussortierte Literatur befasst sich größtenteils mit Generatoren, die weder Multiple-Choice-Aufgaben noch Single-Choice-Aufgaben erzeugen.

2.2.1 Aufgaben generieren

Den Computer zu benutzen, um Aufgaben zu generieren, findet schon lange Anwendung. Insbesondere da das Erstellen von vielen Fragen ein schwieriger und zeitaufwendiger Prozess ist. Wie in der Einleitung bereits ausgeführt, soll eine Aufgabe unter anderem bestimmte Kriterien erfüllen. In den Artikeln *„Multiple-Choice Item Generator for Narrative and Declarative Texts“* [Guillermo et al. 2014] und *„Improve the Output from a MCQ Test Item Generator Using Statistical NLP“* [Foster 2010] wird dementsprechend versucht, diesen Prozess der Aufgabenerstellung von einem Computer übernehmen zu lassen. Hierbei wird ein zugrundeliegender Text von einem Parser analysiert. Der Output sind Sätze, Parse-Bäume, Token, POS-Tags (Parts of Speech) und NER-Tags (Named Entry

Recognition). Diese Sätze werden im folgenden Schritt danach durchsucht, ob sie sich als Fragen eignen und dementsprechend bewertet. Im dritten Schritt werden zu den ausgewählten Fragen entsprechende falsche Antworten erzeugt, die nach ihren semantischen Ähnlichkeitswerten ausgesucht werden. Nach Angabe der Autoren sind die Fragen zu 73,50% grammatikalisch korrekt und zu 68,75% semantisch korrekt (vgl. [Guillermo et al. 2014, 4]).

Da die automatische Fragenerstellung nicht immer korrekt arbeitet und die manuelle Erarbeitung zeitintensiv ist, wird häufiger der Ansatz von Vorlagen verwendet, die verschieden ausgeprägt werden können.

„*The basic idea is that, for example, instead of creating a database of questions like "1+1=2?" we can create a database of question templates like "a+b=c?" together the information that the parameters "a,b,c" will be randomly chosen by the computer, form set "{1,2,3}".*“ [Descalco/Carvalho 2015, 1]

Derartige Generatoren finden oft im mathematischen Bereich Anwendung, da hier leicht Vorlagen erstellt und ausgeprägt werden können. Des Weiteren ist es möglich, nur Rechnungen vorzugeben und das Ergebnis aus den eingesetzten Parametern zu ermitteln. Aus dem oben genannten Beispiel wäre 3+3=6 ebenfalls eine mögliche Ausprägung, obwohl 6 nicht in der Parametermenge enthalten ist. Auch die falschen Antwortalternativen müssen hier nicht mit angegeben werden. Diese können ebenfalls zufällig erzeugt werden. Generell ist der Generator nicht nur auf die niedrige Algebra beschränkt, auch komplexere Rechenaufgaben, wie Matrizen normalisieren oder geometrische Graphen bestimmen, ist dadurch realisierbar (vgl. [Descalco/Carvalho 2015, 2]). Dadurch soll der Mehraufwand, wie Korrigieren und Betreuen, für die Lehrenden möglichst geringgehalten werden. Gleichzeitig kann somit den Studierenden eine Möglichkeit zum Üben innerhalb und außerhalb des Klassenzimmers geboten werden.

Der Artikel *„Multiple-choice questions in Mathematics: automatic generation“* [Nagasaka 2020] betont dabei die gute Anwendbarkeit im Selbststudium. Hierbei wird die Quantität der Aufgaben wichtiger eingestuft als die Qualität, um eine wiederholte Übung und Selbsteinschätzung anbieten zu können. Damit Studierende besser eigenständig ihre Fehler erkennen können, sollten Single-Choice-Aufgaben auf Basis einer Thematik bevorzugt werden. Für die praktische Anwendung wurde ein Tool mit der Programmiersprache Python implementiert, welches die Möglichkeit bietet, die Aufgaben in Moodle (Modular Object-Oriented Dynamic Learning Environment) einzubinden. Die Problematik, dass Moodle MatchJax verwendet und dadurch mathematische Ausdrücke clientseitig gerendert werden,

wird mit einem Moodle KaTeX-Filter gelöst. Dadurch können Ausdrücke bereits vorgerendert verwendet werden, um eine bessere Nutzererfahrung bieten zu können.
Der Artikel befasst sich ausschließlich mit der Generierung von Multiple-Choice-Aufgaben im Mathematikunterricht und zeigt deren Nutzen, das Leitprinzip und das gesamte System auf Basis von Moodle und Python. Die erzeugten Aufgaben, meist rechnerischer Natur, werden in Moodle eingebunden und verwendet.
Kosaku Nagasaka evaluiert auch das Feedback der Studierenden. Hierbei gaben 61,8% an, dass sie das Tool als hilfreich oder als mäßig hilfreich empfanden.
Als Vorteile für den Nutzen der Studenten werden genannt:

- die Wiederholbarkeit von Aufgaben mit verschiedenen zufälligen Werten
- höhere Motivation durch wöchentliche Fragebögen

Als Nachteile werden angeführt:

- das fehlende Feedback bei falscher Beantwortung einer Frage
- ein nicht angepasstes Verhältnis von Aufgabenmenge zu Thematik
- generierte Aufgaben können Zahlen enthalten, die so nicht vom Ersteller beabsichtigt worden sind

2.2.2 Prüfungen generieren

Neben der Erstellung von Aufgaben ist die Generierung ganzer Prüfungen ein relevantes Thema. Dabei soll zum einen der Aufwand der Lehrenden so gering wie möglich gehalten werden und zum anderen verhindert werden, dass Studierende untereinander abschreiben oder zu viel Vorwissen aus Altklausuren einbringen. Im Folgenden werden einzelne Beispiele für verschiedene Ansätze ausgewählt und beschrieben.
Für Prüfungen in Papierform stellt der Artikel *„An automatic generator and corrector of multiple choice tests with random answer keys“* [Assis Zampirolli et al. 2016] eine Lösung vor. Hier werden bereits für eine Prüfung vorgegebene Fragen herangezogen, bei denen die Antwortmöglichkeiten zufällig vertauscht werden. Dabei wird ein Deckblatt für die jeweilige Prüfung erstellt, auf dem die Prüflinge die richtige Antwort ankreuzen können. Eine Bilderkennungssoftware wertet das bearbeitete Deckblatt mit den Ergebnissen aus und bewertet direkt den einzelnen Studierenden. Mit dieser Methodik lässt sich unter anderem verhindern, dass Prüflinge untereinander abschauen können.
Um eine Prüfung zu erhalten, die zufällige Fragen vorsieht, benötigt der Ansatz von Wang et al. [2016, 2 ff.] eine Datenbank mit Fragen, die mit einem Schwierigkeitsgrad versehen wurden. Für die Erstellung der Prüfung wird die Thematik sowie die Anzahl an gewünschten

leichten, mittleren und schweren Fragen vorgegeben. Mit einem heuristischen Algorithmus und der Zielfunktion d (c_1, c_2, c_3) = $|n_1 - c_1| + |n_2 - c_2| + |n_3 - c_3|$ (c Anzahl der ausgewählten Fragen, n Anzahl der gewünschten Fragen) wird zufällig die passende Anzahl an Fragen mit den gewünschten Schwierigkeitsgraden ausgewählt. Die Fragen können mehrere richtige und falsche Antworten enthalten, die in beliebiger Reihenfolge stehen können. Hierbei werden nicht Aufgaben aus parametrisierbaren Templates generiert. Vielmehr werden bereits existierende Fragen zu Prüfungen herausgesucht. Die Unterscheidung der Prüfungen erfolgt durch die Auswahl unterschiedlicher Aufgaben.

Eine Methode, welche die Prüfung der Vorjahre mit einbezieht, bietet der genetische Ansatz von Rahim et al. [2017]. Initial werden hierbei ebenfalls mittels eines zufälligen heuristischen Algorithmus fünf Ausgangsprüfungen erzeugt. Darauf aufbauend werden weitere Prüfungen mit Hilfe eines genetischen Algorithmus generiert. Genauer arbeitet dieser in sechs Schritten: 1) Initialisierung, 2) Bewertung, 3) Auswahl, 4) Kreuzung, 5) Mutation, und 6) Wiederholung ab Schritt zwei, bis die gewünschte Lösung erreicht ist. Bewertet wird nach der Klassifizierung der Bloom'schen Taxonomie. Dabei werden die Level "Wissen" und "Verstehen" als leicht, die Level "Anwendung" und "Analyse" als mittelschwer und die Level "Bewerten" und "Entwickeln" als schwer eingestuft. Bei der Kreuzung und Mutation werden die aktuell ausgewählte und die davor erzeugte Prüfung als Elternteile gesehen, ähnlich wie in der biologischen Vererbung. Jedes Chromosom steht für einen Prüfungssatz und jedes Gen für eine Frage im Prüfungssatz. Wie in Abbildung 2 zu erkennen, wird bei der Kreuzung zufällig ein Kreuzungspunkt gesetzt. Ab diesem Punkt wird die linke Seite des ersten Elternteils und die rechte Seite des zweiten Elternteils als neues Kind verwendet. Bei der Mutation werden einzelne Fragen anhand der Mutationsrate durch neue ersetzt.

Abbildung 2: Beispielhafte Kreuzung [Rahim et al. 2017, 16]

Der parametrisierbare Ansatz von Žitko et al. [2009] bietet eine Möglichkeit, Domänenwissen und Fragestruktur getrennt voneinander betrachten zu können.

Beispielsweise wird hier das Domänenwissen eines Computeraufbaus dargestellt. Dieser hat verschiedene Hardware mit Eingabegeräten, Speicher und ähnliches, verschiedene Software und externe Geräte wie Drucker oder einen Bildschirm. Daneben existieren Frage/Antwort-

Templates. Diese geben die Struktur einer Frage vor: Multiple-Choice-Aufgaben oder Richtig-Falsch-Aufgaben. Ein Beispiel für eine Multiple-Choice-Aufgabe ist in Listing 1 gegeben. Mindestens eine und maximal zwei Antworten der vier gegebenen sind richtig. Als Frage soll ein Wert aus dem Parameter *c* ausgewählt werden. Dieser Parameter wird an den Wertebereich von *Class* gebunden. Als Antwortmöglichkeiten benötigt das Template

```
Type: MultiChoice
MinCorrect: 1
MaxCorrect: 2
Total: 4
Example: Select mouse!
Question pattern: Select ?c
Random Bind: {(?c Class)}
Answer Patterns:
Correct Answer Pattern: {(?x type ?y)(?y subsumes ?c)}
Must Bind: (?x)
Correct Answer Pattern Instance: None
Correct if: countEqual(?x, 0)
Correct Answer Pattern Instance: ?x
Correct if: countGreater(?x, 0)
Incorrect Answer Pattern: {(?x type ?y)(?y subsumes complement((?c))}
Must Bind: (?x)
Incorrect Answer Pattern Instance: None
Incorrect if: countEqual(?x,0)
Incorrect Answer Pattern Instance: ?x
Incorrect if: countGreater(?x, 0)
```

Domänenwissen. Der Parameter *Class* besitzt hierbei Unterkategorien. Darauf folgen Regeln, wann die Frage richtig und wann sie falsch beantwortet wurde. Eine mögliche Ausprägung ist in Abbildung 3 gegeben.

Listing 1: Abfragemuster für die Multiple-Choice-Frage [Žitko et al. 2009, 7]

Select input device:

1. None.
2. HP Optical mouse.
3. Canon Laser printer.
4. DDR RAM 512MB.

Abbildung 3: Beispielhafte Ausprägung [Žitko et al. 2009, 8]

Bei der Erstellung der Prüfung werden nun das Domänenwissen und die Fragestruktur zusammengebracht. Die Thematik des Computers soll beispielsweise mit Single-Choice-Aufgaben, Multiple-Choice-Aufgaben und Richtig-Falsch-Aufgaben abgefragt werden.
Der Algorithmus stellt sicher, dass die gegeben Templates mit dem Domänenwissen vereinbar sind. Darauffolgend wird eine Prüfung mit vorgegebener Schwierigkeit erstellt. Dies ist im gegebenen Computerbeispiel der Tabelle 1 zu entnehmen.

	Question	**Question type and answer type**	**Heaviness category**
1	What is [Class]?	Multiple choice [ClassDescriptions]	Heavy
2	What is [Individual]?	Multiple choice [Class]	Heavy
3	Who has [DatatypeProperty] [Value]	Multiple choice [Individual]	Heavy
4	Select [Class]?	Multiple choice [Individual]	Heavy
5	Does [Class] [ObjectProperty] [Class]?	True/false	Easy
6	Does [Individual] [ObjectProperty] [Individual]?	True/false	Easy
7	Does [Individual] [DatatypeProperty] value?	True/false	Easy
8	Is [Individual] [Class]?	True/false	Easy
9	Is [Class] [Class]?	True/false	Easy
10	Does [Individual] has [DatatypeProperty]	True/false	Easy
11	What is [DatatypeProperty] of [Individual]?	Multiple choice [Value]	Mediate
12	Who has [ObjectProperty] [Individual]	Multiple choice [Individual]	Mediate
13	What is [DatatypeProperty] of [Individual]	Multiple choice [Value]	Mediate
14	What is [ObjectProperty] of [Individual]	Multiple choice [Individual]	Mediate

Tabelle 1: Frage/Antwort-Templates, vereinfachte Beschreibung [Žitko et al. 2009, 9]

Ein großer Vorteil der getrennten Betrachtungsweise ist, dass nur wenige Experten nötig sind, um im Vorfeld das formalisierte Domänenwissen und die Frage-Antwort-Templates zu implementieren. Im Unterrichtsbetrieb können die Lehrenden entscheiden was, wie und wann die Studierenden lernen.
Der Artikel nennt aber selbst auch als großen Nachteil, dass formalisierte Fragen abschreckend sein können, da die Satzstruktur nicht immer ausreichend gut ist.

2.2.3 Lernobjekte erstellen und teilen

Eine weitere Möglichkeit, Varianz in die Aufgaben zu bringen und den Mehraufwand der Lehrenden zu verringern, ist, die einmal erstellten Fragen und Antworten untereinander zu teilen. Es gibt bereits mehrere Lösungen, die eine Datenbank anbieten, auf die über eine grafische Oberfläche zugegriffen werden kann (vgl. [Harley et al. 2013] und [Rjoub et al.

2006]). Innerhalb dieser Oberflächen ist es möglich, neue Fragen zu erstellen und alte hochzuladen. Wenn diese erst einmal im System sind, können andere Lehrende darauf zugreifen und zu den benötigten Thematiken passende Fragen aussuchen. Funktionen, wie sich zufällig eine Anzahl an Fragen für eine Prüfung oder ein Quiz geben zu lassen, sowie diese unterschiedlich anzuordnen, sind ebenfalls möglich.

Der Artikel *„Digitalisierte Hochschuldidaktik: Qualitätssicherung von Prüfungen mit dem E-Assessment-Literacy-Tool EAs.LiT“* [Thor et al. 2017] bietet zusätzlich das Feature an, dass neu erstellte Lernobjekte mit einem Learning-Outcome versehen und nach dem Vier-Augenprinzip auf ihre Qualität überprüft werden können. Mit dem Learning-Outcome wird beschrieben, welche Absicht mit der Frage verfolgt wird, und welches Wissen oder welche Kompetenz abgefragt werden. Dadurch soll die Wahrscheinlichkeit der Wiederverwendung erhöht werden. Besonders da auch hier die Bloom'sche Taxonomie zum Tragen kommt und aus dem vorhandenen Fragepool gleichschwere Prüfungen mit unterschiedlichen Fragen erstellt werden können. Als größter Vorteil wird die *„[…] Vernetzung von Akteuren und Qualitätssicherung von Prozessen“* [Thor et al. 2017, 184] genannt, da verschiedene Einrichtungen unterschiedliche Verwaltungssysteme nutzen. Diese werden durch die gemeinsam genutzte Plattform ersetzt. Förderlich dafür ist der Output des Systems, der in ILIAS (Integriertes Lern-, Informations- und Arbeitskooperations-System) oder Moodle importiert werden kann.

3 Domain Engineering

Domain Engineering, das als eine Entwicklung für Wiederverwendbarkeit beschrieben werden kann, kann wie folgt definiert werden:

„*Domain Engineering is the activity of collecting, organizing and storing past experiences in building systems or parts of systems in a particular domain in the form of reusable assets (i.e. reusable work products), as well as providing an adequate means for reusing these assets (i.e. retrieval, qualification, dissemination, adaption, assembly, and so on) when building new systems.*“ [Czarnecki/Eisenecker 2000, 20]

In der Industrie lässt sich dies mit dem Bau einer Fabrik vergleichen, in der später die fertigen Produkte hergestellt werden. Innerhalb einzelner Produktlinien können beispielsweise Autos vom Kunden individuell angepasst werden, je nachdem welche Merkmale gewünscht sind. Dadurch ist eine hohe Individualisierbarkeit gewährleistet und ein breiteres Marktsegment kann bedient werden. Umgemünzt auf Software-Produktlinien (SPL) sollen softwareintensive Systeme aus einem gemeinsamen Satz von Merkmalen gebaut werden (vgl. [Clements/Northrop 2002, 5]). Dafür werden einzelne Komponenten in Module eingeteilt, die unabhängig voneinander genutzt werden können. Die Implementierung bildet kein fertiges Softwareprodukt, sondern eine Reihe von Einzelteilen. Diese können je nach Anwendungsfall automatisch und manuell kombiniert werden. Durch die erhöhte Flexibilität sollen schneller die Bedürfnisse eines bestimmten Marksegmentes oder einer Aufgabe befriedigt werden (vgl. [Clements/Northrop 2002, 20]).

Domain Engineering lässt sich in Domänenanalyse, Domänendesign und Domänenimplementierung unterteilen. Diese drei Bereiche werden nun mittels der in dieser Arbeit implementierten SPLs näher erläutert.

3.1 Domänenanalyse

Innerhalb der Domänenanalyse soll eine klare Eingrenzung der Domäne erfolgen. Dabei werden Anwendungsbereiche, involvierte Akteure und Ziele identifiziert.

Als Stakeholder ist einerseits der Professur anzuführen. Für den Professor und seine wissenschaftlichen Mitarbeiter soll durch eine Softwarelösung der Arbeitsaufwand bei der Erstellung von Single-Choice-Aufgaben verringert werden. Andererseits können auch die Studierenden profitieren. Eine inhaltlich gleiche Aufgabe kann mehrfach mit unterschiedlichen Werten geübt und wiederholt werden. Durch den deutlich vergrößerten Fragenpool wird dabei auch nicht die Integrität einzelner Fragen beeinträchtigt.

Die Anforderungen an den zu entwickelnden Prototypen sind im Gespräch entstanden. Diese können im Anhang A nachgelesen werden. Die ersten 14 Punkte und Punkt 17 beschäftigen sich mit dem Aufbau der zu übergebenen Schablone und spezifizieren deren Bestandteile. Die dafür entwickelte domänenspezifische Sprache und der genaue Aufbau dieser wird anhand eines Beispiels in Kapitel 4.2 vertieft. In Abbildung 4 können die einzelnen Bestandteile der Schablone grafisch nachvollzogen werden.
Mit dem Punkt 15 werden drei gewünschte Modi beschrieben. Diese werden später die einzelnen Ausprägungen der SPL darstellen. Punkt 16 macht Angaben zum Output und gibt den Verwendungsbereich vor. Die letzten fünf Anforderungen spezifizieren Fehler, auf die die Konfigurationsdatei geprüft werden soll.

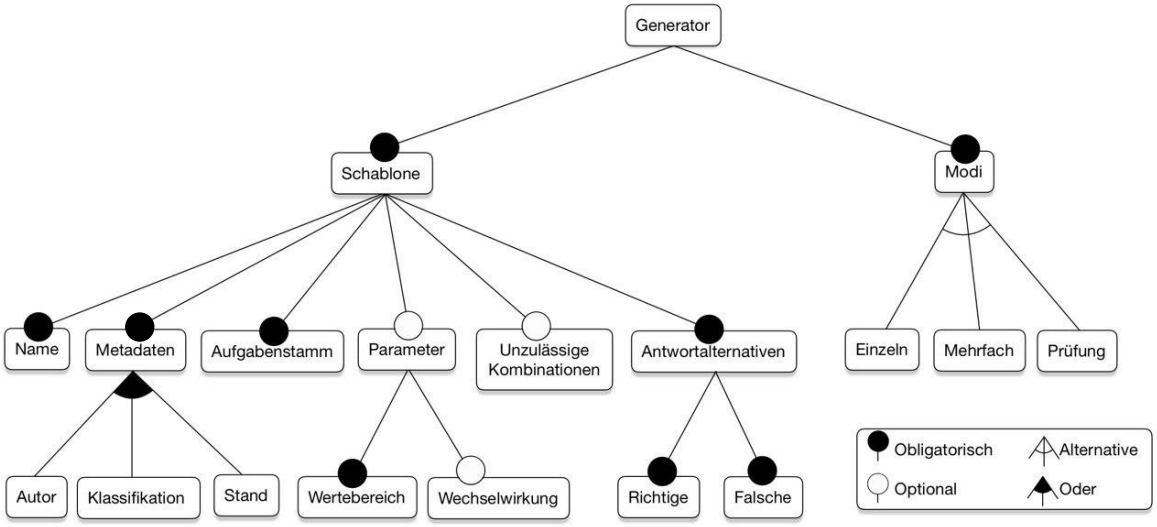

Abbildung 4: Merkmalmodell

3.2 Domänendesign

Anschließend an die Analyse erfolgt das Domänendesign. Hierbei wird die Architektur der SPLs erarbeitet. Diese sollte flexibel sein, da alle Softwareprodukte darauf basieren und keine Einzelsysteme sind (vgl. [Czarnecki/Eisenecker 2000, 28]). Die Gemeinsamkeiten der Domäne bilden die Grundlage zur Einteilung der Software.
In diesem Fall benötigt der Generator unabhängig vom ausgewählten Modus eine oder mehrere Konfigurationsdateien. Hierbei bietet es sich an, diese in das JSON-Format (JavaScript Object Notation) zu überführen, da zahlreiche Bibliotheken zur Verarbeitung existieren, wodurch das weitere Vorgehen erleichtert wird. Ebenfalls muss diese auf Fehler überprüft und die mögliche Aufgabenausprägungen ermittelt werden. Erst nachdem diese

Schritte erfolgt sind, wird eine Unterteilung nach den unterschiedlichen Modi vorgenommen.
Wird ein anderer Programmablauf als der in Kapitel 4.1 beschriebene benötigt, können die Prozessschritte anders kombiniert oder übersprungen werden.

3.3 Domänenimplementierung

Die Implementierung der Domäne beinhaltet die Programmierung der einzelnen Komponenten. Für die Umsetzung wurde die Skriptsprache Python gewählt. Diese verspricht eine einfache Entwicklung, da bereits viele Bibliotheken und Frameworks vorhanden sind, die verwendet werden können. Der Prototyp unterliegt keinen Geschwindigkeitsvorgaben und wird auch in keinem kritischen Bereich eingesetzt. Daher haben eine gute Lesbarkeit und ein knapper Programmierstil höhere Priorität.
Für den Datenaustausch wird das kompakte und gut lesbare JSON-Format verwendet. Es ist programmiersprachenunabhängig, wodurch eine weitere Flexibilität gegeben wird. Unter den einzelnen Komponenten wird als Schnittstelle weitgehend ein JSON-Objekt verarbeitet. Für den Import in ILIAS wird eine XML-Datei (Extensible Markup Language) benötigt. Diese wird am Ende des Erstellungsprozesses generiert.
ILIAS ist eine freie Software für die Bereitstellung einer Lernplattform. Auf dieser können internetbasierte Lehr- und Lernmaterialien (für E-Learning) erstellt und verfügbar gemacht werden. Außerdem bietet das System eine Möglichkeit zur Kommunikation und Kooperation unter Lehrenden und Lernenden, insbesondere durch die Möglichkeit, Prüfungen durchzuführen.
Eine SPL als Generator für Single-Choice-Aufgaben mit einer Anpassung an ILIAS wurde für die Professur Wirtschaftsinformatik an der Universität Leipzig ausgeprägt. Die SPL besitzt den gleichen Merkmalsatz, wie Parameter und Kombinationen (siehe Abbildung 4). Die einzelnen Modi wurden jeweils um die Features der unterschiedlichen Ausgabeformen erweitert.
Zur Erstellung des Prototyps wurde die iterative Methode gewählt, die in „*prototyping tools and techniques*“ [Beaudouin-Lafon/Mackay 2007] beschrieben ist. Hierbei wird das System stetig weiterentwickelt und auch das am Ende abgegebene Produkt ist nicht endgültig. Es handelt sich vielmehr um eine Version, die ausgeliefert wird. Bei jedem weiteren Release kann es zu Änderungen kommen und somit zu neuen Versionen.
Des Weiteren wurde die Strategie „Code and Fix“ angewandt, bei dem zunächst etwas Code geschrieben wird und im Nachhinein dieser auf Fehler überprüft wird. Darunter wird ein

unstrukturiertes Vorgehen verstanden. Hierbei handelt sich zwar um ein unstrukturiertes Vorgehen, was aber akzeptabel ist, da nur ein Entwickler beteiligt ist (vgl. [Zuser et al. 2001, 45]).

4 Tool Beschreibung

Abbildung 5: Grober Programmablauf

Der Generator unterteilt sich im Wesentlichen in vier Bereiche. Zunächst wird eine Konfigurationsdatei eingelesen. Der Parser sucht im String die jeweils benötigten Abschnitte heraus und überführt sie in ein Dictionary. Dieses wird daraufhin überprüft, ob alle Anforderungen erfüllt werden. Als nächstes werden alle erlaubten Antwort-Frage-Kombinationen gebildet und, falls vorhanden, mit den ausgeprägten Parametern kombiniert. Im letzten Schritt werden die Single-Choice-Aufgaben ausgeprägt und in verschiedenen Dateiformaten abgespeichert. Über den Generierungsprozess hinweg wird eine Log-Datei geführt, die die wesentlichen Informationen und potenziellen Fehler zusammenfasst.

4.1 Programmablauf

Es wurden drei verschiedene Verwendungsweisen des Generators umgesetzt (siehe Anhang A, Punkt 15). Die erste erzeugt aus einer Konfigurationsdatei genau eine Aufgabenausprägung. Die zweite erstellt eine gewünschte Anzahl an Aufgabenausprägungen und die dritte generiert aus mehreren Konfigurationsdateien die gewünschte Anzahl an Prüfungen. Zu Beginn wird der Benutzer gebeten, die gewünschte Art des Generators auszuwählen sowie die Dateipfade der Input- und Output-Dateien festzulegen. Der Parser, die Überprüfung der erzeugten JSON-Datei und die Ermittlung aller möglichen Kombinationen läuft bei allen drei Generatoren ähnlich ab. Exemplarisch erfolgt die Erläuterung des Programmablaufs anhand des ersten Generators, da der grundsätzliche Ablauf bis auf wenige spezifische Abweichungen gleich ist.

Parser

Die als Text-Datei vorliegende Konfigurationsdatei wird vom Parser in das JSON-Format überführt. Dadurch lässt sich in den folgenden Schritten leichter damit arbeiten, da Python

über eine JSON-Bibliothek zur Verarbeitung dieser verfügt. Die Umwandlung erfolgt mittels String-Suche. Die Reihenfolge und das Vorhandensein der Schlüsselwörter sind daher entscheidend. Näheres zum Aufbau der Konfigurationsdatei wird in Kapitel 4.2 beschrieben. Die umgewandelte Konfigurationsdatei dazu ist im Anhang Listing 5: „Beispielhafte Konfigurationsdatei im JSON-Format“ angegeben.

Prüfen der JSON-Datei

Die JSON-Schablone wird anhand folgender Kriterien untersucht:

- ist die Schablone nicht leer und entspricht auch einem Dictionary?
- sind alle Schlüsselwörter enthalten?
- gibt es genügend Antwortalternativen?
- sind die Parameter richtig aufgebaut, werden definiert und verwendet und besitzen einen Wertebereich?

Ist eines dieser Kriterien nicht erfüllt, so wird eine entsprechende Fehlermeldung ausgegeben und der weitere Programmverlauf abgebrochen. Eine Aufschlüsselung, wie die Fehler zu interpretieren sind, ist in Kapitel 4.4 gezeigt.

Kombinatorik

Auf Grundlage der korrekten JSON-Schablone werden alle Kombinationen ermittelt, die diese zulässt. Jede Ausprägung eines Parameters wird mit den verbleibenden Ausprägungen der anderen Parameter verbunden. Dabei wird darauf geachtet, dass die Wechselwirkungen eingehalten werden. Genauso werden alle Kombinationen der falschen Antworten gebildet und mit den richtigen verknüpft. Unzulässige Antwortalternativpaare werden danach aussortiert. Diese beiden Mengen werden ebenfalls wieder miteinander kombiniert. Die neue Menge kann zu viele Elemente enthalten, die zu mehreren gleichen Ausprägungen führen können. Wenn ein Parameter in einer Antwort-Alternative nicht enthalten ist, kann dieser auch keinen Einfluss darauf nehmen. Beispielhaft ausgedrückt (dazu Abbildung 6 betrachten): Der zweite Parameter kommt bei der Antworten-Kombination 1, 5, 6, 8 nicht vor. Die Ausprägungen, die *$PARTWO* besitzt, sind somit irrelevant, da *value2* in der Aufgabe nicht eingesetzt werden kann. Nur der erste Parameter kann unterschiedliche Varianten der Antworten-Kombination erzeugen. Würden diese allerdings dennoch mit den Ausprägungen des ersten Parameters kombiniert werden, gäbe es doppelte Single-Choice-Aufgaben.

Die Lösungsmenge wird in eine Liste geschrieben, die wiederum Listenelemente mit jeweils zwei Einträgen enthalten. Im Ersten sind die Antwortalternativen und im Zweiten sind die Parameter mit den expliziten Ausprägungen enthalten.

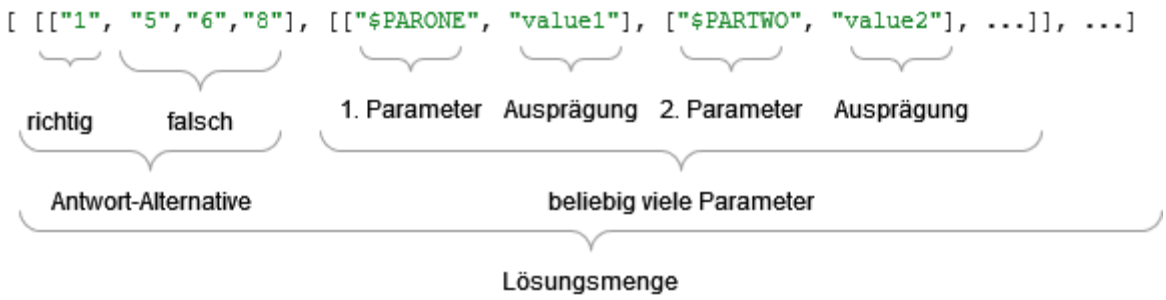

Abbildung 6: Aufbau der Lösungsmenge mittels Kombinatorik

Ersetzung

Im Schritt der Ersetzung wird die Benutzereingabe überprüft und wenn nötig angepasst. Danach kommen die Ergebnisse der vorherigen Schritte zusammen. Die in der Lösungsmenge enthaltenen Antwortalternativen dienen als Schlüssel für die JSON-Schablone, um die richtigen Werte zu erhalten. Für die angegebenen Antworten werden die Parameter nun mit den passenden Ausprägungen mittels einfacher String-Ersetzung ersetzt.

Der Output

Die Anforderung an den Output des Generators ist, dass die erzeugte Datei sich problemlos in ILIAS importieren lässt (siehe Anhang A, Punkt 16). Zu Beginn wurde dafür der „ILIAS XmlWriter“ von Wiltrud Kessler, 5.11.13 verwendet. Dieser übernimmt eine Text-Datei und setzt die Frage und Antworten in ein XML-Format ein. Daher werden bei den ersten beiden Modi zusätzlich noch die erzeugten Aufgaben in Textform ausgegeben.

Bei zunehmender Komplexität des Programms, insbesondere bei der Erstellung unterschiedlicher Prüfungen, zeigte sich die Notwendigkeit eine angepasste Form zu verwenden. Diese bietet ebenfalls die Möglichkeit, alle Aufgaben in eine XML-Datei zu übernehmen, wodurch nur eine Datei in ILIAS hochgeladen werden muss.

Der Dateiname setzt sich aus drei Bestandteilen zusammen: einer sequenziell hochgezählten Nummer, dem Übersetzungszeitpunkt und dem Namen. Zuerst wird jede Datei fortlaufend nummeriert. Die Nummerierung entfällt, wenn es sich um eine Datei handelt, in der alle Aufgaben vereint abgelegt sind. Darauf folgt der Zeitpunkt, zu welchem der Generator gestartet wurde, um eine Art Versionierung zu erzeugen. Der letzte Bestandteil ist der Name der Aufgabe beziehungsweise der Prüfung, gefolgt von der Dateiendung.

4.2 Die Konfigurationsdatei

Dieser Abschnitt thematisiert den korrekten Aufbau der Konfigurationsdatei mittels der domänenspezifischen-Sprache. Die Punkte 1-14 & 17 der Anforderung werden behandelt (siehe Anhang A). Punkt 1 gibt das Format der Text-Datei für die Konfigurationsdatei fest vor.

```
1.  ***NAME***
2.  Definition einer Variable mit Initialisierung
3.  ***AUTOR***
4.  Ulrich Eisenecker
5.  ***STAND***
6.  17.5.2001
7.  ***VERSION***
8.  1.0
9.  ***KOMMENTAR***
10. ###LEHRSTOFF, KLASSIFIKATION, EINSTUFUNG
11. Kapitel "Variablen", Anwendungsfrage, mittel
12. ***PARAMETER***
13. $TYPE
14. $NAME
15. $VALUE
16. ***WERTEBEREICH***
17. ###$TYPE:
18. int *** double *** std::string
19. ###$NAME:
20. a *** b *** c *** myVar *** some_var
21. ###$VALUE:
22. 42 *** 99 *** 0 *** 0.0 *** 3.14 *** 120.0 *** "OMG"s *** "Hi,
    there" *** "Good "s + "point
23. ***WECHSELWIRKUNG***
24. $TYPE int $VALUE 42 *** 99 *** 0
25. $TYPE double $VALUE 0.0 *** 3.14 *** 120.0
26. $TYPE std::string $VALUE "OMG"s *** "Hi, there" *** "Good "s +
    "point
27. ***AUFGABENSTAMM***
28. How do you define the variable $NAME of type $TYPE and initialize
    it with value $VALUE?
29. What is the correct statement?
30. ***RICHTIG***
31. $TYPE $NAME { $VALUE };
32. $TYPE $NAME = $VALUE;
33. ***FALSCH***
34. $TYPE $NAME; $NAME = $VALUE;
35. $NAME as $TYPE = $VALUE;
36. $TYPE $NAME { }; $NAME = $VALUE;
37. $NAME = $TYPE($VALUE);
38. auto $NAME = new $TYPE($VALUE);
39. ***KOMBINATIONEN***
40. 3 exclude 5
41. ***ENDE***
```

Listing 2: Beispiel einer Konfigurationsdatei

Diese Konfigurationsdatei wird dem Generator übergeben (Listing 2). Dementsprechend müssen die einzelnen Bestandteile kenntlich gemacht werden, um an späterer Stelle vom Parser verarbeitet werden zu können. Die Zeilen 10, 17, 19, 21 und 40 werden mit ###

eingeleitet. Diese Zeilen sind jeweils Kommentare und werden nicht weiter vom Parser beachtet. *Name*, *Autor*, *Stand*, *Version*, *Kommentar*, *Parameter*, *Wertebereich*, *Wechselwirkung*, *Aufgabenstamm*, *richtig*, *falsch*, *Kombinationen* und *Ende* bilden die Schlüsselwörter für die einzelnen Abschnitte. Diese müssen mit *** beginnen und enden, sowie großgeschrieben werden. Sollte ein Abschnitt nicht verwendet werden, muss er trotzdem an der richtigen Stelle genannt werden und leer bleiben. Die Reihenfolge darf nicht verändert werden. *Name, Autor* und *Kommentar* enthalten die jeweilige Information, die der Titel nahe legt. *Stand* und *Version* folgen diesem Schema, allerdings sind sie nur in der Konfigurationsdatei enthalten. In der ausgeprägten Frage tauchen sie nicht mehr auf.

Die Bestandteile von Zeile 12 bis 41 bezwecken eine variierende Ausprägung der Konfigurationsdatei. Für eine Single-Choice-Aufgabe können beliebig viele Parameter definiert werden (Z. 12-15). Sie müssen mit einem $-Zeichen beginnen und dürfen nur Buchstaben von A-Z enthalten. Die definierten Parameter dürfen in den Abschnitten *Name, Aufgabenstamm, richtig* und *falsch* verwendet werden.

Wenn Parameter definiert sind, muss für jeden ein eigener Wertebereich vergeben werden (Z. 16-22). Die einzelnen Wertebereiche werden jeweils in einer Zeile angegeben und mit *** getrennt. Wichtig ist: Die Reihenfolge der zuvor definierten Parameter bestimmt auch die Zuordnung der Wertebereiche. Hilfreich ist es hierbei, die Parameter als Kommentar eine Zeile darüber zu schreiben (Z. 17, 19, 21).

Wenn Parameter definiert wurden, können auch Wechselwirkungen unter diesen angegeben werden (Z. 23-26). Im oben gezeigten Beispiel sollen die Werte zu den jeweiligen Typen passen: Bei Integer nur Ganzzahlen, bei Double nur Gleitpunktzahlen und bei String nur Zeichenketten. Dafür wird zuerst der bestimmende Parameter gefolgt von einem Wert genannt. Darauf folgt der abhängige Parameter mit den entsprechenden Werten, wieder getrennt mit ***. Die Zeile 24 kann wie folgt in natürliche Worte gefasst werden: „Wenn für den Parameter $TYPE der Wert int ausgeprägt wird, dann kann der Parameter $VALUE nur noch mit den Werten 42, 99 oder 0 ausgeprägt werden."

Der *Aufgabenstamm* (Z. 27-29) enthält die Frage. *Richtig* (Z. 30-32) und *falsch* (Z. 33-38) bestehen aus den jeweils richtigen beziehungsweise falschen Antwortalternativen. Jede Zeile ist eine Antwort. Es muss mindestens eine richtige und drei falsche Alternativen geben.

Der Abschnitt *Kombinationen* umfasst die unzulässigen Antwortkombinationen (Z. 39, 40). Diese werden in Zahlen angegeben, beginnend bei eins bis hin zur gesamten Antwortanzahl. Die Antworten werden intern durchnummeriert. Im obigen Beispiel schließt die falsche Antwort in Zeile 34 die in Zeile 36 aus und umgekehrt (dient nur zur Veranschaulichung, semantisch schließen sich die Antworten nicht aus).

4.3 Besonderheiten

An dieser Stelle werden einige Besonderheiten beleuchtet, die bei näherer Untersuchung des Prototyps hilfreich sind. Die Module verfügen über Debug-Variablen. Werden diese zu Beginn auf `True` gesetzt, sind der Programmablauf, Rückgabewerte und einzelne Prozessschritte nachvollziehbar.

In der Übersetzungseinheit string_cleaning.py ist die Variable `max_file_name_length` auf 150 gesetzt. Diese bewirkt, dass der Dateiname die Länge von 150 Zeichen nicht überschreitet. Bei der Entwicklung des Prototyps wurden zu langen Dateinamen vom Betriebssystem abgeschnitten und verkürzt gespeichert. Bei einem erneuten Versuch, die Dateien zu öffnen, konnten diese nicht mehr unter dem zulangen Namen gefunden werden. Sollte der Fehler auf einer anderen Umgebung weiterhin auftauchen, kann an dieser Stelle die Länge angepasst werden. Der Wert sollte nicht zu klein gewählt werden. Ansonsten ist die Unterscheidung durch Hochzählen der Dateien nicht mehr gegeben, da bei einer Verkürzung drei Punkte angehängt werden. Zum Beispiel erlaubt der Wert fünf nur Dateien bis 9 (1_....txt), der Wert sechs nur Unterscheidungen bis 99 (99_....txt). Werden mehr Dateien geschrieben, werden keine weiteren neuen Namen erzeugt, sondern die alten überschrieben.

Die folgenden beiden Besonderheiten beziehen sich auf Listing 5: „Beispielhafte Konfigurationsdatei im JSON-Format“, welches im Anhang wiederzufinden ist. Zuerst wird ein Hinweis auf die Bezeichnung der Zugriffe und auf die Wechselwirkungen gegeben (Z. 21-33). Der bestimmende Parameter wird als *key1* (Z. 22) und seine Ausprägungen als *key2* (Z. 23, 26, 29) bezeichnet. Der abhängige Parameter wird *key3* genannt. Diese Begrifflichkeit wird bei der Kombinatorik und in der Log-Datei häufig verwendet.

Die Schlüsselwörter *Kombinationen* und *Counting Combinations* (Z. 46-52) bilden im Grunde die gleiche Information ab. Da ausschließende Antwortalternativen sich bidirektional bedingen, werden diese auch so abgelegt. Für den Nutzer ist allerdings eine vereinfachte Darstellung angenehmer zu handhaben.

4.4 Die Log-Datei

Folgend wird auf die Log-Datei näher eingegangen. Die Datei wird während der Prozessschritte aktualisiert und gibt dem Benutzer Aufschluss über diese. Die Punkte 18-22 der Anforderung werden behandelt (siehe Anhang A). Es werden zwei Log-Dateien betrachtet, Listing 3: „Beispielhafte Log-Datei“ ohne Fehler in der Konfigurationsdatei und

Listing 6: „Log-Datei mit Fehlerreport“. Zweitere ist aus Platzgründen im Anhang wiederzufinden.

```
1. {
2.       "Parser": "All components are included",
3.        "JSON": {
4.            "dict_type": true,
5.            "file_is_not_empty": true,
6.            "all_components": true,
7.            "enough_correct_answers": 2,
8.            "enough_wrong_answers": 5,
9.            "parameters_are_built_correctly": true,
10.           "each_parameter_is_used_at_least_once": true,
11.           "each_parameter_used_is_defined": true,
12.           "each_parameter_in_interaction_is_defined": true,
13.           "interaction_are_achievable": true,
14.           "parameters_have_value_set": true
15.       },
16.       "Combinatorics": {
17.           "amount_pos_combi": 630,
18.           "possible_scopes_combination": [...],
19.           "possible_question_combination": [...],
20.           "pos_qs_combination": [...]
21.       },
22.       "Template is expressible": true,
23.       "Input is achievable": true
24.  }
```

Listing 3: Beispielhafte Log-Datei

In der Log-Datei sind die ersten drei großen Schritte wieder zu finden: Parser, Prüfung und Kombination. Im Parser-Segment wird festgehalten, ob alle Bestandteile einer korrekt aufgebauten Konfigurationsdatei enthalten sind (Listing 3 Z. 2). Ist dies nicht der Fall, werden alle fehlenden Bestandteile aufgenommen und der weitere Prozess wird abgebrochen (Listing 6 Z. 2-4). Die darauffolgenden Einträge der Log-Datei wären dann mit NULL versehen. Listing 6 ist daher als ein aggregiertes Beispiel anzusehen. Die einzelnen gefundenen Fehler sind allerdings so in unterschiedlichen Dateien wiederzufinden.

Der Schritt der Prüfung und somit die Fehler der übergebenen Konfigurationsdatei sind im JSON-Tag zusammengefasst. „`dict_type`“, „`file_is_not_empty`“, „`all_components`“ sind noch einmal mit aufgenommen und finden Anwendung, wenn der Parser davor nicht verwendet wird, sondern direkt eine Konfigurationsdatei im JSON-Format übergeben wird. Listing 6 Z. 8-13 zeigen den gleichen Fehler erneut, dass die Komponente *Stand* nicht in der Konfigurationsdatei enthalten ist. „`dict_type`“ wird `false`, wenn die übergebene Datei nicht dem Typ eines Dictionary entspricht. „`file_is_not_empty`“ wird `false`, wenn die übergebene Datei leer ist.

Die folgenden beiden Abschnitte spiegeln wider, ob genügend richtige (eine) oder falsche (drei) Antwortalternativen angegeben wurden. Die entsprechende Anzahl wird festgehalten.

Die definierten Parameter müssen mit einem $-Zeichen beginnen und dürfen nur die Großbuchstaben A-Z enthalten. Ist dies nicht der Fall, wird ein entsprechender Fehler ausgegeben (siehe Listing 6 Z. 18-25).
Die letzten vier Tags decken die Anforderungen 18-22 ab (siehe Anhang A). Wenn ein Parameter nicht verwendet wird, gibt es eine Fehlermeldung (Listing 6 Z. 26-31). Bei Wörtern, die wie ein Parameter aufgebaut aber nicht zuvor definiert worden sind, wird das entsprechende Wort und sein Fundort in „`each_parameter_used_is_defined`“ gespeichert (Listing 6 Z. 32-39).
Wenn nicht definierte Parameter in den *Wechselwirkungen* verwendet werden, wird ebenfalls ein Fehler ausgegeben und Ort und Parameter werden vermerkt (Listing 6 Z. 40-45). Der Ort in Zeile 43 (*key1*) benennt den bestimmenden Parameter. Der genaue Aufbau ist im Kapitel 4.3 Besonderheiten zu finden.
Eine Wechselwirkung ist nicht erfüllbar, wenn ein Parameter von mehreren anderen Parametern bestimmt wird. In Listing 6 Z. 46-51 wird *$PARONE* von *$PARTHREE* und *$PARTWO* beeinflusst und eine entsprechende Fehlermeldung ausgegeben.
Hat ein Parameter keinen Wertebereich, wird ein Fehler ausgegeben (Listing 6 Z. 52-56). Kommt es zu diesem Fehler, ist allerdings genau zu prüfen, ob es auch wirklich der angegebene Parameter ist, für den kein Wertebereich definiert worden ist. Durch die feste Reihenfolge bei der Zuweisung der Wertemenge zu den Parametern kann bereits eine verrutschte Zeile diesen Fehler hervorrufen.
„`Combinatorics`“ umfasst keine Fehler. Hier werden die Rückgabewerte aus dem Schritt der Kombination festgehalten. Die Einträge in den Zeilen 59 bis 61 (Listing 6) beinhalten normalerweise viele Einträge, wurden aber aus Platzgründen verkürzt dargestellt.
Wenn mittels der Konfigurationsdatei mindestens eine Single-Choice-Aufgabe erzeugt werden kann, ist das Template ausprägbar (Listing 3 Z. 22).
Wird das Programm in Modus 1 ausgeführt, wird zusätzlich geprüft, ob die Nutzereingabe die gewünschten Antwortalternativen und Parameter-Kombinationen anhand der Konfigurationsdatei erfüllbar sind (Listing 3 Z. 23). Dieser Tag ist bei den anderen beiden Modi nicht vergeben. Bei der Ausführung des Programms in Modus 3 kommen noch fünf weitere Eigenschaften hinzu: 1) Der Name der Prüfung, 2) die Anzahl der erstellten Prüfungen, 3) die Anzahl der Aufgaben pro Prüfung und 4) ob Aufgaben wiederholt werden müssen, da es für eine Konfigurationsdatei nicht genügend unterschiedliche Ausprägungen gibt. Des Weiteren werden 5) die Kommentare der einzelnen Schablonen erneut mit abgelegt. Diese bieten einen Überblick über die enthaltenen Kapitel und den Schwierigkeitsgraden auf oberer Ebene.

4.5 Testing

Testen ist Teil der laufenden Prozessaktivitäten Verifizierung und Validierung. Mittels der Verifikation wird untersucht, ob das System richtig gebaut wurde und somit den Anforderungen entspricht. Bei Validierung hingegen wird hinterfragt, ob das richtige System entwickelt und die Erwartungen des Kunden erfüllt wurden.

Wie die Anforderungen (siehe Anhang A) umgesetzt wurden, kann in den vorherigen Kapiteln nachvollzogen werden. Mit Listing 6 wurde ein Beispiel gezeigt, welches viele Fehlerreports enthält. Um sicherzustellen, dass weitere Fehler auch korrekt aufgedeckt werden, wurde eine Vielzahl von Unittests geschrieben. Für eine bessere Nachvollziehbarkeit untersucht jeweils ein Test eine Fehlerquelle. Unter der Zuhilfenahme des Frameworks pytest können die Tests automatisch ausgeführt werden. Wird mit der Kommandozeile in den Ordner *testing* navigiert, können dort mit dem Befehl *pytest* alle Tests ausgeführt werden. Zwei Tests werden nun exemplarisch vorgestellt.

Die Konfigurationsdatei aus dem Anhang (Listing 7: „Konfigurationsdatei im JSON-Format für die Überprüfung“) besitzt einen Fehler. Im Bereich der Wechselwirkungen (Z. 17-32) wird der abhängige Parameter *$PARONE* von *$PARTWO* und *$PARTHREE* bestimmt. Dieses Verhalten kann nicht abgebildet werden, wodurch ein Fehler ausgegeben und in der Log-Datei vermerkt wird (Listing 8: „Erwartete Log-Datei“). Der gezeigte Aufbau der Unittests ist weitgehend gleich. Einer Funktion wird eine Konfigurationsdatei übergeben und der Rückgabewert wird verglichen mit der zuvor festgelegten Erwartung.

Das gleiche gilt für die Funktion, die für die Errechnung aller möglichen Aufgaben zuständig ist. Unter der Annahme, dass ein Template keine Wechselwirkungen und keine unzulässigen Kombinationen besitzt, kann die Anzahl der sich unterscheidenden Aufgaben wie folgt berechnet werden:

Annahme: keine Wechselwirkungen, keine unzulässigen Kombinationen

r Anzahl richtiger Antworten

f Anzahl falscher Antworten

$$r \times \left(\frac{f!}{3! \times (f-3)!}\right) \times Ausprägungen\ pro\ Parameter$$

Abbildung 7: Anzahlberechnung der Lösungsmenge

Der zweite Multiplikator ist der Binomialkoeffizient über Drei, da nur drei falsche Antwortmöglichkeiten ausgewählt werden können. Diese werden mit den einzelnen

Ausprägungen der Parameter und der Anzahl der richtigen Antworten multipliziert. Sind keine Parameter vorhanden, muss in die Formel eine Eins eingesetzt werden.

Im Beispiel aus dem Listing 9: „Konfigurationsdatei im JSON-Format für den Kombinatorik-Test“ sind drei richtige, fünf falsche Antwortalternativen und drei Parameter enthalten. Diese haben fünf, zwei und drei Ausprägungsmöglichkeiten. Wären keine Wechselwirkungen und keine unzulässigen Kombinationen angegeben, könnten 30 Antwortmöglichkeiten und 30 Parameterausprägungskombinationen gebildet werden. Zusammen könnten 900 sich unterscheidende Single-Choice-Aufgaben erstellt werden.

$$3 \times \left(\frac{5!}{3! \times (5-3)!}\right) \times (5 \times 2 \times 3) = 900$$

Durch die Einschränkung der Wechselwirkungen unter den Parametern entfallen 300 Kombinationen. Um die Gesamtzahl zu errechnen, muss der hintere Teil der Formel geändert werden. Es werden nicht mehr alle Parameterausprägungen multipliziert, sondern die Ausprägungen der abhängigen mit denen der unabhängigen. 30 Antwortmöglichkeiten multipliziert mit 20 (10×2) Parameterausprägungen ergeben 600 Möglichkeiten (Abbildung 8, mittlerer Abschnitt). Durch die unzulässigen Antwortkombinationen sind nur noch neun unterschiedliche möglich $(20 \times 9 = 180)$ (Abbildung 8, linker Abschnitt).

Bei genauerer Betrachtung fällt auf, dass nicht in jeder Antwortkombination alle Parameter vorkommen. Bei der Auswahl von (2, 4, 5, 6) ist der *$PARTHREE* nicht enthalten. Die Auswahl von [*$PARONE*, int1; *$PARTWO*, double1; *$PARTHREE*, char1] und [*$PARONE*, int1; *$PARTWO*, double1; *$PARTHREE*, char3] sind mit der Wechselwirkung der Parameter vereinbar. Dadurch dass sich aber ausschließlich der dritte Parameter verändert, würden zwei identische Aufgaben ausgeprägt werden. Daher muss die Zahl der sich unterscheidenden ausgeprägten Aufgaben weiter gesenkt werden $20 \times 9 \neq 180$ (Abbildung 8, rechter Abschnitt). Werden diese Fälle nicht mit betrachtet, sind 170 Elemente in der Lösungsmenge enthalten.

Legende:

Entallen durch die Wechselwirkung
Entfallen durch unzulässige Kombinationen
Entfallen durch fehlende Parameter

Nur Antwort-Kombinationen	Nur die Wechselwirkung			Antwortkombination ohne dritten Parameter			
	$PARONE	*$PARTWO*	*$PARTHREE*		*$PARONE*	*$PARTWO*	*$PARTHREE*
('1', '4', '5', '6'),	int1	double1	char1	2, 4, 5, 6	int1	double1	char1
('1', '4', '5', '7'),	int1	double1	char2		int1	double1	char2
('1', '4', '5', '8'),	int1	double1	char3		int1	double1	char3
('1', '4', '6', '7'),	int1	double2	char1		int1	double2	char1
('1', '4', '6', '8'),	int1	double2	char2		int1	double2	char2
('1', '4', '7', '8'),	int1	double2	char3		int1	double2	char3
('1', '5', '6', '7'),	int2	double1	char1		int2	double1	char1
('1', '5', '6', '8'),	int2	double1	char2		int2	double1	char2
('1', '5', '7', '8'),	int2	double1	char3		int2	double1	char3
('1', '6', '7', '8'),	int2	double2	char1		int2	double2	char1
('2', '4', '5', '6'),	int2	double2	char2		int2	double2	char2
('2', '4', '5', '7'),	int2	double2	char3		int2	double2	char3
('2', '4', '5', '8'),	int3	double1	char1		int3	double1	char1
('2', '4', '6', '7'),	int3	double1	char2		int3	double1	char2
('2', '4', '6', '8'),	int3	double1	char3		int3	double1	char3
('2', '4', '7', '8'),	int3	double2	char1		int3	double2	char1
('2', '5', '6', '7'),	int3	double2	char2		int3	double2	char2
('2', '5', '6', '8'),	int3	double2	char3		int3	double2	char3
('2', '5', '7', '8'),	int4	double1	char1		int4	double1	char1
('2', '6', '7', '8'),	int4	double1	char2		int4	double1	char2
('3', '4', '5', '6'),	int4	double1	char3		int4	double1	char3
('3', '4', '5', '7'),	int4	double2	char1		int4	double2	char1
('3', '4', '5', '8'),	int4	double2	char2		int4	double2	char2
('3', '4', '6', '7'),	int4	double2	char3		int4	double2	char3
('3', '4', '6', '8'),	int5	double1	char1		int5	double1	char1
('3', '4', '7', '8'),	int5	double1	char2		int5	double1	char2
('3', '5', '6', '7'),	int5	double1	char3		int5	double1	char3
('3', '5', '6', '8'),	int5	double2	char1		int5	double2	char1
('3', '5', '7', '8'),	int5	double2	char2		int5	double2	char2
('3', '6', '7', '8')	int5	double2	char3		int5	double2	char3

Abbildung 8: Teilabschnitt der Lösungsmenge

Die gesamte Aufschlüsslung der wegfallenden Elemente, sowie die Lösungsmenge sind in der Exceldatei „Kombinationen“ gegeben.

5 Benutzung

Der Prototyp enthält einen Generator mit drei Modi. Er kann über die Kommandozeile ausgeführt werden. Dafür muss der Benutzer sich im Ordner, in dem auch der Prototyp liegt, befinden und den Befehl „`python ./Bachelorarbeit/src/__main__.py`“ ausführen. Der Benutzer wird nun aufgefordert, einen Modus auszuwählen. Dieser spiegelt die Verwendung des jeweiligen Generators wider. Modus 1 prägt eine spezifische Aufgabe aus, Modus 2 eine gewisse Anzahl und Modus 3 generiert aus gegebenen Konfigurationsdateien eine gewünschte Anzahl an Prüfungen (siehe Anhang A, Punkt 15). Darauffolgend muss angegeben werden, wo die Konfigurationsdatei innerhalb des Prototyps liegt. Normalerweise befindet diese sich im Ordner *Config_txt*. Ein Dialog führt den Benutzer durch die weiteren Abfragen. Um die Benutzung des Tools zu erleichtern, werden dem Nutzer Beispielwerte angezeigt. Ebenso wurden Standardwerte für den Output festgelegt. Sind diese nicht gewünscht, können sie verändert werden. Wird keine Eingabe getätigt, wird weiterhin der Vorgabewert verwendet. Sind alle Werte gesetzt, werden sie noch einmal angezeigt und der entsprechende Generator gestartet.

Im Ordner *Pruefungen_OUTPUT* werden die zusammengefassten Aufgaben und Prüfungen für die weitere Verwendung abgespeichert. Vor jedem Durchlauf wird der Inhalt dieses Ordners gelöscht, um Verwechslungen zu vermeiden. Die Ausführung in Modus 1 und 2 speichert die Aufgaben zusätzlich im Ordner *Question_import*. Die bei jeder Ausführung neu erstellte Konfigurationsdatei im JSON-Format ist im Ordner *Config_json* zu finden.

Der erste Modus gibt deren ungetestete Version direkt nach dem Parser aus. Dies kann genutzt werden, um die Text-Datei einfacher auf Fehlern zu prüfen. Nach der Verwendung des Generators können mittels Modus -1 die Dateien der Output-Ordner und die Log-Dateien gelöscht werden. Eine grafische Darstellung, wann welcher Ordner zum Einsatz kommt, ist im Anhang B, Abbildung 9: Ordnerstruktur zu finden.

Als zusätzliche Eigenschaft kann der Dialog der Benutzereingabe übersprungen werden. Dafür müssen bei der Ausführung des Programms sieben Argumente übergeben werden. Dabei können die Standardwerte nicht angepasst werden. Das erste Argument legt den 1) Modus fest. Im zweiten Argument wird der 2) Ort der Konfigurationsdatei angegeben. Die folgenden beiden geben 3) Antwortalternativen und 4) Parameterausprägungen an. Hierbei ist auf die Unterteilung zu achten: Kommata zwischen Parameter und Wert, Semikolon zwischen den Parameter-Wert-Paaren. Der fünfte Parameter gibt 5) die Anzahl der Aufgaben an, die ausgeprägt werden sollen. Wird eine Zahl kleiner Eins verwendet, werden alle Aufgaben ausgeprägt, die möglich sind. Mit dem vorletzten Argument wird die gewünschte

6) Anzahl an Prüfungen bestimmt. Die Angabe über den Ort des Inputs kann hierbei entfallen, weil er nicht geändert werden kann. Dieser befindet sich im Ordner *Pruefung_INPUT*. Der siebte Parameter gibt den festen 7) Bestandteil der Prüfungen vor. In Listing 4 sind vier Beispiele an Argumenten gegeben, die zeigen, wann und wo welches Argument gesetzt werden muss.

```
python Bachelorarbeit/src/__main__.py
Vier Beispiele von sieben Übergabeparameter
-1 "" "" "" "" "" ""
1 ./Bachelorarbeit/Config_txt/config_bsp2.txt "6,7,1,4"
                "$VALUE, 0.0; $TYPE, double; $NAME, myVar" "" "" ""
2 ./Bachelorarbeit/Config_txt/config_bsp1.txt "" "" "10" "" ""
3 "" "" "" "" "50" "meine_Prüfung"
```

Listing 4: Prototyp ohne Dialog ausführen

Für einen schnellen Einstieg mit eigenen Werten in einer Schablone, ist eine leere Konfigurationsdatei im Anhang gegeben (Listing 10: „Leere Konfiguration“).

6 Fazit und Evaluation

In diesem Abschnitt werden die Bedrohungen der Validität diskutiert. Außerdem wird betrachtet, ob der Prototyp für den gedachten Zweck benutzt werden kann.

Einige Aspekte werden von dem Prototyp nicht berücksichtigt. Bei der Überprüfung wird nicht abgefragt, ob zu viele Wertebereiche angegeben wurden. Außerdem werden nicht alle Möglichkeiten der Wechselwirkungen abgedeckt. Es wird beispielsweise nicht erneut betrachtet, wenn ein Parameter zwei oder mehrere weitere bestimmt. Gleichermaßen können keine Ketten gebildet werden: *$PARONE* bestimmt *$PARTWO* und dieser bestimmt *$PARTHREE*.

Dennoch erreicht der Generator für Single-Choice-Aufgaben die Ziele. Ein wesentliches Potenzial ist die Prüfungserstellung mit sich unterscheidenden Aufgaben bei dennoch gleichbleibendem Schwierigkeitsgrad.

Auf den ersten Blick ist das Erstellen von Single-Choice-Aufgaben mittels einer Schablone schwierig. Allerdings reichen bereits zwei richtige, fünf falsche Antwortalternativen und ein Parameter mit vier Ausprägungen aus, um 80 unterschiedliche Aufgaben generieren zu können. Bei einer Kursgröße von beispielsweise 60 Studierenden ist das bereits hinreichend. Mit einem relativen geringen Aufwand kann eine große Anzahl an unterschiedlichen Aufgaben generiert werden.

7 Ausblick

Der Prototyp stellt kein fertiges Produkt dar und lässt sich noch um einige Features erweitern. Im Folgenden sollen einige Erweiterungsmöglichkeiten vorgestellt werden.

Zuallererst können die Punkte aus Kapitel 6 mit betrachtet und in den Generator mit implementiert werden.

Bei der Erstellung mehrerer Prüfungen wird jede Prüfung in einer eigenen XML-Datei abgelegt. Bei einer Anzahl von 500 Prüfungen ist der händische Import in ILIAS zeitaufwändig und fehleranfällig. An dieser Stelle kann geprüft werden, welche Voraussetzungen gegeben sein müssen, um diesen zu vereinfachen. Dies ist beispielsweise möglich durch das Erstellen einer großen XML-Datei oder eines komprimierten Ordners, der alle einzelnen Prüfungen enthält.

Des Weiteren werden einzubindende Grafiken nicht mitverarbeitet. Die Aufgaben können Mark-Ups beinhalten, die auf den Speicherort dieser referenzieren. Die Bilder können allerdings nicht vom Generator verarbeitet werden. Es müsste ein Modul entwickelt werden, welches den Unterschied von Konfigurationsdateien und Bildern erkennt. Der Ordner für den Input könnte die Konfigurationsdateien und die benötigten Bilder enthalten. Grafiken oder anderweitige Dateien würden weiterhin nicht vom Generator verarbeitet werden. Diese könnten dennoch mit in den Output Ordner kopiert werden für einen einfach Import in ILIAS.

Die Bearbeitung von Konfigurationsdateien innerhalb der beschriebenen domänenspezifische-Sprache in Textformat kann leicht unübersichtlich werden. Für deren leichtere Handhabung wäre eine grafische Oberfläche (GUI) sinnvoll. Innerhalb der GUI könnte fortlaufend die Konfigurationsdatei auf Fehler überprüft und die möglichen Kombinationen könnten angezeigt werden. Dadurch könnte der aktuelle Schritt des Parsers entfallen. Die Eingaben des Nutzers könnten direkt von der GUI verarbeitet und als JSON-Objekt zurückgegeben werden.

Eine Generatorausprägung für Multiple-Choice-Aufgaben ist mit dem vorhandenen Wissen realisierbar. Hierbei wird zusätzlich die Anzahl der richtigen beziehungsweise falschen Antworten benötigt sowie die Angabe, wie viele Antwortalternativen in einer Aufgabe enthalten sein sollen. Danach wären Anpassungen in allen Modulen nötig. Die meisten müssten im Abschnitt durchgeführt werden, der für die Ermittlung der Kombinationen zuständig ist, da nicht nur mehrere richtige Antworten, sondern auch eine variable Anzahl an falschen Antworten mit betrachtet werden müsste.

Außerdem wäre eine Datei für den Import in andere Systeme denkbar. Beispielsweise für die Moodle-Plattform könnte der Generator im zweiten Modus eine mögliche Ergänzung darstellen. Hierfür müsste nur die ausgegebene XML-Datei an das XML-Schema von Moodle angepasst werden.
Schlussendlich wäre eine Datenbankanbindung ebenfalls eine mögliche Erweiterung. Hierbei können Konfigurationsdateien der Single-Choice-Aufgaben abgespeichert und für die spätere Erstellung von Prüfungen oder Self-Assessment-Aufgaben verwendet werden.
Eine Anbindung an das System EAs.LiT (vorgestellt in Kapitel 2.2.3) ist denkbar.

Literaturverzeichnis

[Assis Zampirolli et al. 2016] Assis Zampirolli, F., Batista, V.R., Quilici-Gonzalez, J.A., An automatic generator and corrector of multiple choice tests with random answer keys, 2016 IEEE Frontiers in Education Conference (FIE), 2016, S. 1–8.

[Beaudouin-Lafon/Mackay 2007] Beaudouin-Lafon, M., Mackay, W.E., Prototyping Tools and Techniques, The Human-Computer Interaction Handbook, CRC Press, 2007, S. 1043–1066.

[Bloom 1956] BLOOM, B.S., Taxonomy of educational objectives: The classification of educational goals, in: Cognitive domain 1956.

[Burr et al. 2015] Burr, W., Schmidt, X., Frohwein, T., Ausgewählte Ergebnisse zur Qualitätsoptimierung der Lehre an der Universität Stuttgart, Universität Stuttgart, 2015.

[Clements/Northrop 2002] Clements, P., Northrop, L., Software product lines, 2002.

[Czarnecki/Eisenecker 2000] Czarnecki, K., Eisenecker, U.W., Components and Generative Programming, 2001.

[Descalco/Carvalho 2015] Descalco, L., Carvalho, P., Using parameterized calculus questions for learning and assessment, 2015 10th Iberian Conference on Information Systems and Technologies (CISTI), IEEE, 2015, S. 1–5.

[Foster 2010] Foster, R.M., Improve the Output from a MCQ Test Item Generator Using Statistical NLP, 2010 10th IEEE International Conference on Advanced Learning Technologies, IEEE, 2010, S. 368–369.

[Guillermo et al. 2014] Guillermo, D.J.J., Llona, L.N., Tolentino, M.L., Domingo, R.B., Sagum, R.A., Multiple-Choice Item Generator for Narrative and Declarative Texts, 2014.

[Harley et al. 2013] Harley, Z., Harley, A.W., Harley, E.R., Sharing a wizard for E-learning computer programming, 2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE), IEEE, 2013, S. 161–164.

[Nagasaka 2020] Nagasaka, K., Multiple-choice questions in Mathematics:automatic generation, revisited, 2020 (2020), S. 1–15.

[Okoli/Schabram 2010] Okoli, C., Schabram, K., A Guide to Conducting a Systematic Literature Review of Information Systems Research, in: SSRN Electronic Journal (2010).

[Rahim et al. 2017] Rahim, T.N.T., Aziz, Z., Rauf, R.H., Shamsudin, N., Automated exam question generator using genetic algorithm, 2017 IEEE Conference on e-Learning, e-Management and e-Services (IC3e), IEEE, 2017, S. 12–17.

[Rjoub et al. 2006] Rjoub, A., Tall, B., Sharou, N., Mardeeni, L., A Novel Multi-Forms Multiple Choice Editor Exam Tool Based on HTML Website, 2006 7th International Conference on Information Technology Based Higher Education and Training, IEEE, 2006, S. 854–869.

[Thor et al. 2017] Thor, A., Pengel, N., Wollersheim, H., Digitalisierte Hochschuldidaktik: Qualitätssicherung von Prüfungen mit dem E-Assessment-Literacy-Tool EAs.LiT, in: 179-184 (2017).

[Wang et al. 2016] Wang, J., Chen, F., Tai, D.W., Chen, J., An Automatic Test Generator for Enhancing the Technolgy Profciency of Senior High School Students, 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), IEEE, 2016, S. 273–276.

[Zimmaro 2004] Zimmaro, D.M., Writing good multiple-choice exams, 2004.

[Žitko et al. 2009] Žitko, B., Stankov, S., Rosić, M., Grubišić, A., Dynamic test generation over ontology-based knowledge representation in authoring shell, in: Expert Systems with Applications, 36 (2009) 4, S. 8185–8196.

[Zuser et al. 2001] Zuser, W., Bifel, S., Grechenig, T., Köhle, M., Software-Engineering: Mit UML und dem Unified Process, 2001.

Anhang

Anhang A: Anforderungen

1. Eine Schablone für eine MC MC-Aufgabe liegt als Textdatei vor.
2. Sie kann an beliebiger Stelle Kommentare enthalten.
3. Ein Kommentar wird am Zeilenanfang mit der Sequenz ### eingeleitet.
4. Der erste Abschnitt enthält den Namen der Aufgabenschablone.
5. Der zweite Abschnitt enthält den Autor.
6. Der dritte Abschnitt enthält die Information zum Stand (Datum) und zur Version
7. Nun folgt ein Abschnitt, der Informationen zur Aufgabe enthält, nämlich einen Verweis auf den zugrunde liegenden Lehrstoff, die Klassifikation als FAKTEN-, ANWENDUNGS – oder TRANSFERFRAGE und die Einstufung als LEICHT, MITTEL oder SCHWER.
8. Die Textdatei mit der Schablone für die MC-Aufgabe enthält einen Abschnitt, welcher die Parameter definiert.
9. Pro Parameter gibt es einen weiteren Abschnitt, der den Wertebereich des Parameters definiert.
10. Schließlich kann es beliebig viele Abschnitte geben, von denen jeder einzelne genau eine Wechselwirkung zwischen den Parametern beschreibt.
11. Schließlich kommt ein Abschnitt, der den Aufgabenstamm enthält.
12. Dann kommen mindestens vier Abschnitte, die zulässige Antwortalternativen enthalten. Mindestens eine Antwortalternative muss als RICHTIG gekennzeichnet sein, mindestens drei Antwortalternativen müssen als FALSCH gekennzeichnet sein.
13. Darauf folgen beliebig viele Abschnitte, die beschreiben, welche der Antwortalternativen NICHT miteinander kombiniert werden dürfen.
14. Der Aufgabenstamm und die Antwortalternativen dürfen beliebige Markup-Informationen zur Formatierung und Verweise auf einzubindende Grafiken enthalten.
15. Der Generator kann auf drei Weisen verwendet werden:
    1. Der Benutzer wählt für jeden Parameter einen zulässigen Wert aus. Herbei werden beständig die bestehenden Wechselwirkungen geprüft, so dass keine fehlerhafte Spezifikation erfolgen kann. Nach Abschluss der Spezifikation erstellt der Generator die entsprechende konkrete Aufgabe als Datei mit dem angegebenen Namen.

2. Der Benutzer gibt an, wie viele Aufgaben der Generator erzeugen soll. Der Generator prüft, ob die gewünschte Anzahl erzeugt werden kann und erstellt anschließend genauso viele sequenziell nummerierte Aufgaben als Dateien, wie der Benutzer angegeben hat. Der Benutzer gibt den festen Bestandteil des Dateinamens ebenfalls vor. Alle Aufgaben müssen sich entweder hinsichtlich der verwendeten Werte für die Parameter oder hinsichtlich der ausgewählten Antwortalternativen oder hinsichtlich beidem unterscheiden.
3. Der Benutzer gibt an, wie viele Prüfungen generiert werden sollen. Der Generator erstellt von daraufhin für jedes Template eine unterschiedliche Fragenversion pro Prüfung. Wenn sich nicht genug Versionen ausprägen lassen, soll eine zufällige Version für die Prüfung gewählt werden. Pro Prüfung werden alle Aufgaben dann in eine Ausgabedatei geschrieben, sodass so viele Dateien wie angegebene Prüfungen herauskommen. Die Dateien sollen sequenziell nummeriert und mit dem festen Bestandteil versehen werden.

16. Die erzeugten Dateien sollen sich problemlos in ILIAS importieren lassen.
17. Die Kennzeichnung von Parametern, Werten, Wechselwirkungen und allen weiteren Bestandeilen soll so erfolgen, dass Überschneidungen mit herkömmlichem Text und Mark-Ups sehr unwahrscheinlich sind.
18. Wenn Parameter definiert sind, aber nicht verwendet werden, soll eine entsprechende Fehlermeldung ausgegeben werden.
19. Wenn Parameter verwendet werden, die nicht definiert sind, soll eine entsprechende Fehlermeldung ausgegeben werden.
20. Wenn Wechselwirkungen definiert sind, die nicht definierte Parameter verwenden, soll eine entsprechende Fehlermeldung ausgegeben werden
21. Wenn Wechselwirkungen definiert sind, die nicht erfüllbar sind, soll eine entsprechende Fehlermeldung ausgegeben werden.
22. Wenn leere Wertemengen für Parameter definiert sind, soll eine entsprechende Fehlermeldung ausgegeben werden.

Anhang B: Abbildungen

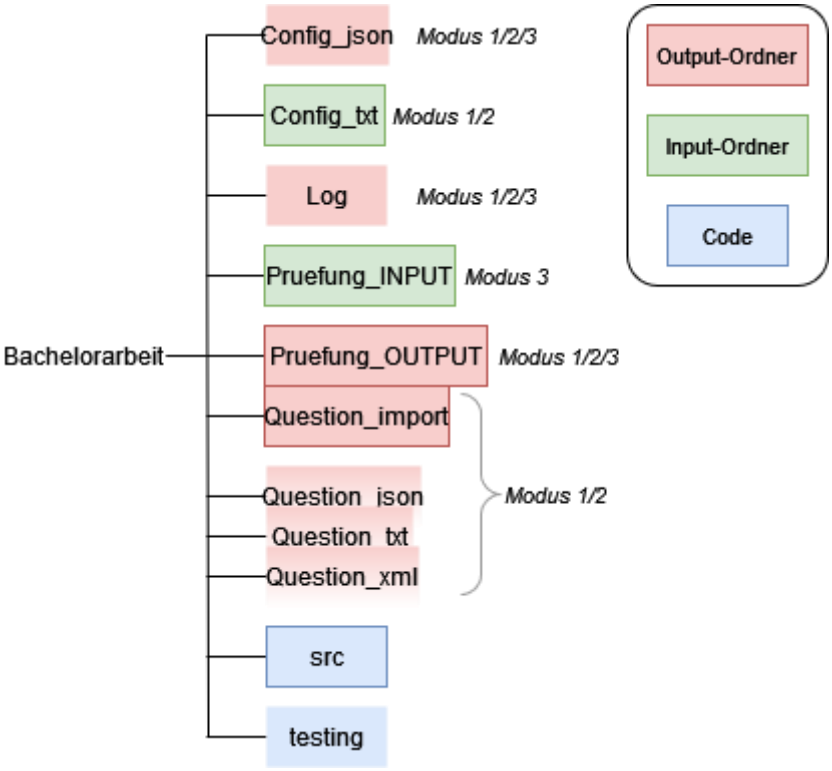

Abbildung 9: Ordnerstruktur

Anhang C: Listings

```
1. {
2.       "Name": "Definition einer Variable mit Initialisierung",
3.       "Autor": "Ulrich Eisenecker",
4.       "Stand": "17.5.2001",
5.       "Version": "1.0",
6.       "Kommentar": "Kapitel \"Variablen\", Anwendungsfrage, mittel",
7.       "Parameter": {
8.            "1": "$TYPE",
9.            "2": "$NAME",
10.           "3": "$VALUE"
11.      },
12.      "Wertebereich": {
13.          "$TYPE": [
14.              "int",
15.              "double",
16.              "std::string"
17.          ],
18.          "$NAME": ["a", "b", "c", "myVar", "some_var"],
19.          "$VALUE": ["42", "99", "0",
                        "0.0", "3.14", "120.0",
                        "\"OMG\"s", "\"Hi,there\"", "\"Good\"s+\"point"]
20.      },
21.      "Wecheslwirkung": {
22.          "$TYPE": {
23.              "int": {
24.                  "$VALUE": ["42", "99", "0"]
25.              },
26.              "double": {
27.                  "$VALUE": ["0.0", "3.14", "120.0"]
28.              },
29.              "std::string": {
30.                  "$VALUE": ["\"OMG\"s", "\"Hi,there\"",
                                "\"Good\"s+\"point"]
31.              }
32.          }
33.      },
34.      "Aufgabenstamm": "How do you define the variable $NAME of type
   $TYPE and initialize it with value $VALUE?\nWhat is the correct
   tatement?",
35.      "Richtige Antworten": {
36.          "1": "$TYPE $NAME { $VALUE };",
37.          "2": "$TYPE $NAME = $VALUE;"
38.      },
39.      "Falsche Antworten": {
40.          "3": "$TYPE $NAME; $NAME = $VALUE;",
41.          "4": "$NAME as $TYPE = $VALUE;",
42.          "5": "$TYPE $NAME { }; $NAME = $VALUE;",
43.          "6": "$NAME = $TYPE($VALUE);",
44.          "7": "auto $NAME = new $TYPE($VALUE);"
45.      },
46.      "Kombinationen": {
47.          "3": ["5"],
48.          "5": ["3"]
49.      },
50.      "Counting combinations": {
51.          "3": ["5"]
52.      }
53.  }
```

Listing 5: Beispielhafte Konfigurationsdatei im JSON-Format

```
{
        "Parser": {
            "***STAND***": "is not included in the template"
        },
        "JSON": {
            "dict_type": false,
            "file_is_not_empty": false,
            "all_components": {
                "value": false,
                "hits": {
                    "Stand": ["is not in the config dict"]
                }
            },
            "enough_correct_answers":
                {"value": false, "hits": {"correct_answers": [0]}},
            "enough_wrong_answers":
                {"value": false, "hits": {"wrong_answers": [1]}},
            "parameters_are_built_correctly": {
                "value": false,
                "hits": {
                    "PARONE": ["starts not with \"$\""],
                    "$par2": ["is not alpha", "is not uppercase"],
                    "$parthree": ["is not uppercase"]
                }
            },
            "each_parameter_is_used_at_least_once": {
                "value": false,
                "hits": {
                    "$PARTHREE": ["is not used"]
                }
            },
            "each_parameter_used_is_defined": {
                "value": false,
                "hits": {
                    "Aufgabenstamm": ["$PARFOUR"],
                    "Richtige Antworten 1": ["$PARFOUR"],
                    "Falsche Antworten 8": ["$PARFOUR"]
                }
            },
            "each_parameter_in_interaction_is_defined": {
                "value": false,
                "hits": {
                    "not defined key1": ["$PARFOUR"]
                }
            },
            "interaction_are_achievable": {
                "value": false,
                "hits": {
                    "$PARONE": ["$PARTHREE", "$PARTWO"]
                }
            },
            "parameters_have_value_set": {
                "value": false,
                "hits": {
                    "$PARTHREE": ["has no value range"]
        }}}},
        "Combinatorics": {
            "amount_pos_combi": 0,
            "possible_scopes_combination": [...],
            "possible_question_combination": [...],
            "pos_qs_combination": [...]},
        "Template is expressible": false,
        "Input is achievable": "The ANSWER Input can not be fulfilled"
    }
```

Listing 6: Log-Datei mit Fehlerreport

```
{
        "Name": "key3 wird mehrfach verwendet",
        "Autor": "Jonathan Thiemann",
        "Stand": "01.12.2021",
        "Version": "1.0",
        "Kommentar": "Test, Faktenfrage, leicht ",
        "Parameter": {
            "1": "$PARONE",
            "2": "$PARTWO",
            "3": "$PARTHREE"
        },
        "Wertebereich": {
            "$PARONE": ["int1", "int2", "int3", "int4", "int5"],
            "$PARTWO": ["double1", "double2"],
            "$PARTHREE": ["char1", "char2", "char3"]
        },
        "Wecheslwirkung": {
            "$PARTHREE": {
                "char1": {
                    "$PARONE": ["int1", "int2"]},
                "char2": {
                    "$PARONE": ["int3", "int4", "int5"]},
                "char3": {
                    "$PARONE": ["int1", "int3", "int5"]}
            },
            "$PARTWO": {
                "double1": {
                    "$PARONE": ["int1", "int2"]},
                "double2": {
                    "$PARONE": ["int1", "int3", "int5"]}
           }
        },
        "Aufgabenstamm": "$PARONE $PARTWO $PARTHREE Lorem ipsum",
        "Richtige Antworten": {
            "1": "1 $PARONE $PARTWO $PARTHREE Lorem ipsum",
            "2": "2 $PARONE $PARTWO $PARTHREE Lorem ipsum",
            "3": "3 $PARONE $PARTWO $PARTHREE Lorem ipsum"
        },
        "Falsche Antworten": {
            "4": "4 $PARONE $PARTWO $PARTHREE Lorem ipsum",
            "5": "5 $PARONE $PARTWO $PARTHREE Lorem ipsum",
            "6": "6 $PARONE $PARTWO $PARTHREE Lorem ipsum",
            "7": "7 $PARONE $PARTWO $PARTHREE Lorem ipsum",
            "8": "8 $PARONE $PARTWO $PARTHREE Lorem ipsum"
        },
        "Kombinationen": {},
        "Counting combinations": {}
    }
```

Listing 7: Konfigurationsdatei im JSON-Format für die Überprüfung

```
{
        "dict_type": true,
        "file_is_not_empty": true,
        "all_components": true,
        "enough_correct_answers": 3,
        "enough_wrong_answers": 5,
        "parameters_are_built_correctly": true,
        "each_parameter_is_used_at_least_once": true,
        "each_parameter_used_is_defined": true,
        "each_parameter_in_interaction_is_defined": true,
        "interaction_are_achievable": {
            "value": false,
            "hits": {
                "$PARONE": [
                    "$PARTHREE",
                    "$PARTWO"
                ]
            }
        },
        "parameters_have_value_set": true
}
```

Listing 8: Erwartete Log-Datei

```
1. {
2.       "Name": "TEST Parameter nicht in allen Fragen, Wechselwirkung,
   Kombinationen",
3.       "Autor": "Jonathan Thiemann",
4.       "Stand": "01.12.2021",
5.       "Version": "1.0",
6.       "Kommentar": "Test, Faktenfrage, leicht ",
7.       "Parameter": {
8.           "1": "$PARONE",
9.           "2": "$PARTWO",
10.          "3": "$PARTHREE"
11.      },
12.      "Wertebereich": {
13.          "$PARONE": ["int1", "int2", "int3", "int4", "int5"],
14.          "$PARTWO": ["double1", "double2"],
15.          "$PARTHREE": ["char1", "char2", "char3"]
16.      },
17.      "Wecheslwirkung": {
18.          "$PARTHREE": {
19.              "char1": {
20.                  "$PARONE": ["int1", "int2"]},
21.              "char2": {
22.                  "$PARONE": ["int3", "int4", "int5"]},
23.              "char3": {
24.                  "$PARONE": ["int1", "int2", "int3", "int4", "int5"]}
25.          }
26.      },
27.      "Aufgabenstamm": "$PARONE $PARTWO Lorem ipsum",
28.      "Richtige Antworten": {
29.          "1": "1 Lorem ipsum",
30.          "2": "2 $PARONE $PARTWO Lorem ipsum",
31.          "3": "3 $PARONE $PARTWO $PARTHREE Lorem ipsum"
32.      },
33.      "Falsche Antworten": {
34.          "4": "4 Lorem ipsum",
35.          "5": "5 $PARONE Lorem ipsum",
36.          "6": "6 $PARONE $PARTWO Lorem ipsum",
37.          "7": "7 $PARONE $PARTWO $PARTHREE Lorem ipsum",
38.          "8": "8 $PARONE $PARTWO $PARTHREE Lorem ipsum"
39.      },
40.      "Kombinationen": {
41.          "1": ["4", "7"],
42.          "4": ["1", "8"],
43.          "7": ["1"],
44.          "3": ["8"],
45.          "8": ["3", "4", "5"],
46.          "5": ["8"]
47.      },
48.      "Counting combinations": {
49.          "1": ["4", "7"],
50.          "3": ["8"],
51.          "4": ["8"],
52.          "5": ["8"]
53.      }
54.  }
```

Listing 9: Konfigurationsdatei im JSON-Format für den Kombinatorik-Test

```
***NAME***
###
***AUTOR***
###
***STAND***
###
***VERSION***
###
***KOMMENTAR***
###LEHRSTOFF, KLASSIFIKATION, EINSTUFUNG
###
***PARAMETER***
###
***WERTEBEREICH***
###
***WECHSELWIRKUNG***
###
***AUFGABENSTAMM***
###
***RICHTIG***
###
***FALSCH***
###
***KOMBINATIONEN***
###
***ENDE***
```

Listing 10: Leere Konfigurationsdatei