Institut für Rechnergestützte Ingenieursysteme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis Nr. 180

Entwicklung eines Sprachverarbeitungs- und Bewertungssystems zur automatischen Vorkorrektur

Robin Reutter

Studiengang:	Informatik B.Sc.
Prüfer:	Univ-Prof. Hon-Prof. Dr. Dieter Roller
Projektkoordinator:	Dipl.-Inf. Felix Baumann, M. Sc. Julian Eichhoff
begonnen am:	24. Oktober 2014
beendet am:	24. April 2015
CR-Klassifikation:	D.2.2, D.2.3, I.2.7, I.7.1

Kurzfassung

Das Korrigieren schriftlicher Aufgaben nach Musterlösung stellt für eine lehrende Person – beispielsweise den Lehrer an einer Schule, den Tutor einer Studentengruppe oder den Mitarbeiter einer Universität – häufig eine mühsame und zeitintensive Aufgabe dar. Das Kontrollieren der Ergebnisse und Abgleichen mit der Musterlösung kann insbesondere bei sich wiederholenden und eindeutigen Lösungen sehr ermüdend sein. Die in Anspruch genommene Zeit könnte zur Vergabe von Lernhinweisen oder dem intensiveren Befassen mit der eigentlichen Korrektur weitaus besser genutzt werden.

In diesem Zusammenhang existiert der Wunsch nach einem automatisierten System, das die Vorkorrektur übernimmt und die Lehrkraft entlastet. Korrekt gelöste Aufgaben könnten im Voraus von dem System mit voller Punktzahl bewertet werten, im Falle von Unklarheit könnte die Lehrkraft entscheiden, wie die eingesendete Lösung zu bewerten ist. Bei der riesigen Anzahl verschiedenartiger Aufgaben müssten Kategorien für Aufgabentypen und Lösungsmöglichkeiten erstellt werden.

Diese Arbeit stellt ein Modell speziell für die „Aufgaben der Informatik 1“ vor. Die Aufgaben wurden unter Berücksichtigung bereits existierender Ansätze in Kategorien eingeteilt. Ein System auf Grundlage von C# wurde zur automatischen Erstellung und Vorkorrektur der Aufgaben entwickelt und evaluiert.

Abstract

Correcting written exercises according to its sample solution is in many cases a laborious and time-consuming task for a teaching person – e.g. a teacher at a school, a tutor of a student group or a scientific researcher at a university. Especially regarding repetitive and clear solutions checking the result and comparing it to the sample solution can be very exhausting. This time spent could instead be used quite better for giving hints concerning the solution or intensifying attending the actual correction.

In this context the request for an automated system taking care of pre-correction and reducing work for the teaching person arises. Exercises being solved correctly could be marked with maximum points – in case of unclarity the teaching person could decide how to rate the solution. For the huge amount of different exercise types, categories for exercise types and solution possibilities would have to be created.

This work suggests a prototype specifically addressing “Aufgaben der Informatik 1”. The exercises were categorized in consideration of already existing work. A system based on C# for automatic pre-correction of these exercises was developed and evaluated.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Ein wichtiger Bestandteil des Lernprozesses ist das regelmäßige Anwenden des Erlernten in Form von Übung. Regelmäßiges Üben wird weitgehend als erfolgsentscheidendes Kriterium für den Lernerfolg angesehen. In vielen Studiengängen ist die regelmäßige, erfolgreiche Teilnahme an Übungen während des Semesters Pflicht, um zu Prüfungen zugelassen zu werden. [1] Innerhalb der Übungen müssen Aufgaben bis zu einem gewissen Grad erfolgreich gelöst werden.

Entsprechend dieser Anforderungen benötigt es Lehrende, die sich mit der teilweise sehr zeitaufwändigen Korrektur von Übungen befassen. Je umfangreicher die Übung ist, desto umfangreicher ist auch die darauffolgende Korrektur. Bei großen Lerngruppen und umfangreichen Aufgaben wird die Korrektur somit ein erheblicher Aufwand. Insbesondere das Durchlesen korrekter Antworten nimmt sehr viel Zeit in Anspruch. Diese Zeit könnte beispielsweise weitaus besser zur eigentlichen Korrektur falscher Lösungen oder der intensiveren Vergabe von Lernhinweisen genutzt werden.

Vor diesem Hintergrund ist eine Unterstützung wünschenswert, die unnötige Korrekturarbeit abnimmt. Es gibt zahlreiche Systeme in vielen verschiedenen Gebieten, die Lehrenden helfen, Aufgaben zu stellen und zu korrigieren. Vor Allem an Hochschulen finden diese Lernsysteme, größtenteils als E-Learning-Plattformen - auch in Kombination mit Verwaltungssystemen und weiteren Funktionalitäten - hohen Gebrauch. Sie bieten zahlreiche Möglichkeiten, das Stellen und Korrigieren von Aufgaben wird durch Automatismen vereinfacht und weniger zeitaufwändig. Dadurch kann wiederum die Qualität der eigentlichen Korrektur durch die Lehrkraft angehoben werden.

In einem Rundumschlag wurden in dieser Bachelorarbeit verschiedene Übungsaufgaben einer Aufgabensammlung zunächst in Kategorien unterteilt und daraufhin unterschiedliche Wege gesucht, diese Aufgaben mithilfe eines Programms automatisiert zu korrigieren.

Einige der oben genannten Systeme wurden auf ihre Fähigkeiten hin untersucht beim Korrigieren von Aufgaben zu unterstützen. Außerdem wurden wissenschaftliche Beiträge und Arbeiten, die sich mit der automatisierten Korrektur von Aufgaben beschäftigten, untersucht.

Das schlussendliche Ziel dieser Arbeit war der Entwurf und die Entwicklung einer Softwarekomponente, die Antworten in digitaler Form zu gegebenen Aufgabenstellungen bewertet und vorkorrigiert.

Das System sollte dabei auch eine Struktur zum Format, der digitalen Einsendeaufgaben und der Korrekturergebnisse, vorgeben. In einer späteren Arbeit soll es möglich sein auf Grundlage dieses Systems weitere Methoden zur automatischen Vorkorrektur zu entwickeln und mit einzubinden. Insbesondere maschinelle Lernverfahren, die die automatische Vorkorrektur im System trainieren, könnten bei der automatischen Vorkorrektur von großem Nutzen sein.

1.1 Struktur der Bachelorthesis

Zunächst werden in Kapitel 2 die Grundlagen und Rahmenbedingungen, die bei der Entwicklung des Programms wichtig waren, erläutert und festgelegt. Im 3. Kapitel wird auf

unterschiedliche existierende Systeme und wissenschaftliche Beiträge sowie wissenschaftliche Bereiche, die für diese Arbeit relevant waren, eingegangen. Kapitel 4 erläutert das Konzept dem das entworfene System zugrunde liegt, die Realisierung des Konzepts wird im 5. Kapitel vorgestellt. Das entworfene System wird daraufhin in Kapitel 6 analysiert und die daraus gewonnenen Erfahrungen evaluiert. Abschließend wird in Kapitel 7 die Schlussfolgerung der Arbeit formuliert, sowie ein Ausblick auf mögliche künftige Verbesserungen gegeben.

2 Grundlagen & Klassifikation der Aufgabentypen

Die Arbeit beschäftigt sich mit dem Aufgabenbereich der Vorlesung „Grundlagen der Informatik I“. Um präziser veranschaulichen und erklären zu können, wurde passend zum Aufgabenbereich in dieser Arbeit das Szenario „Übungsgruppen an einer Universität“ gewählt. Bei der lehrenden Person, die die Aufgabenstellung vorgibt und die Aufgaben korrigiert, wird vom „Tutor“ – bei der bearbeitenden Person, die die Aufgaben löst und einreicht, vom „Studenten“ gesprochen. Bei den bearbeiteten Aufgaben handelt es sich um „Abgaben“.

Es gilt zunächst die verschiedenen Aufgaben zu klassifizieren, um bereits existierende Arbeiten gezielt suchen und einen Lösungsansatz für das Aufgabenfeld schaffen zu können.

Die Aufgaben der Bereichs „Grundlagen der Informatik I“ werden in dieser Arbeit zunächst in zwei Hauptkategorien, „Algorithmische Aufgaben“ und „Textaufgaben“ unterteilt und dann im Detail weiter gegliedert. Zu jeder Aufgabenstellung gehört auch eine Musterlösung die Grundlage für die Bewertung der Aufgabe darstellt.

2.1 Algorithmische Aufgaben

Hierunter werden zunächst alle Aufgaben verstanden, die das Anwenden eines Algorithmus erfordern.

Definition Algorithmus:

> *„Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift, die von einem mechanisch oder elektronisch arbeitenden Gerät bzw. auch von einem Menschen durchgeführt werden kann. Aus der Präzision der sprachlichen Darstellung des Algorithmus muss die Abfolge der einzelnen Verarbeitungsschritte eindeutig hervorgehen. […] Beispiele für Algorithmen sind z.B. Vorschriften zum Addieren, Subtrahieren oder Multiplizieren von Zahlen. […] Ein Algorithmus legt fest, wie Eingabedaten schrittweise in Ausgabedaten umgewandelt werden.“* [2]

Verdeutlichen lässt sich das am Beispiel „Berechnen Sie den größten gemeinsamen Teiler zweier Zahlen nach dem euklidischen Algorithmus“. Zur Berechnung des größten gemeinsamen Teilers mithilfe des euklidischen Algorithmus dividiert man zunächst die beiden Zahlen mit Rest. Im nächsten Rechenschritt werden der Divisor des ersten Rechenschritts zum Dividend und der Rest zum Divisor. Diese Rechenschritte wiederholt man solange, bis sich der Rest „0“ ergibt, der Divisor des letzten Rechenschritts ist dann der größte gemeinsame Teiler (Abbildung 1).

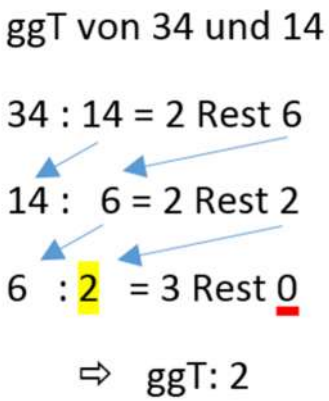


Abbildung 1: Berechnung ggT zweier Zahlen

Bei dieser Aufgabe gilt es für das Programm zu überprüfen, ob die Person die beiden Schritte korrekt angewandt und somit verstanden hat. Etwaige Rechenfehler sollten erkannt werden und in die Bewertung mit einfließen, aber nicht zur kompletten Ablehnung des Ergebnisses führen. Fehler beim Verständnis des Algorithmus sollten ebenfalls erkannt und angezeigt werden. Diese Fehler sind entsprechend schwerwiegender als Rechenfehler und sollten somit stärker gewichtet in die Bewertung miteinfließen.

Der Aufgabensteller muss im Allgemeinen also die Möglichkeit haben, dem System mitzuteilen, nach welchen Vorschriften korrigiert werden muss.

2.2 Textaufgaben

Hierunter werden zunächst alle Aufgaben verstanden, die in einer (freien) Textform und nicht mittels Anwenden eines vorgegeben Algorithmus mit eindeutig definierten Schritten, wie in 2.1, zu lösen sind. Textaufgaben können in zahlreichen Formen auftreten weswegen es einer weiteren Unterteilung bedarf. Die große Schwierigkeit bei einer Textaufgabe tritt auf, je offener Fragestellung und Antwortmöglichkeiten der Aufgabe sind. Je freier die Antwortmöglichkeit, desto komplexer ist es für das System diese auf Korrektheit zu überprüfen. Bei natürlicher Sprache sind die verschiedenartigen Antwortmöglichkeiten nahezu unbegrenzt, weswegen Methoden gefunden werden müssen, eine Antwort in natürlicher Sprache maschinell zu verarbeiten, um sie bewerten zu können (Computerlinguistik).

Für das Aufgabenfeld „Grundlagen der Informatik I“ wurden in dieser Arbeit folgende Kategorien für Textaufgaben erstellt.

2.2.1 Strukturierte Textaufgabe

Unter „Strukturierten Textaufgabe“ werden Textaufgaben verstanden, die keinen oder nur sehr wenig Spielraum für Antwortmöglichkeiten lassen. Hierzu gehören einfache Begriff- und Zuordnungsfragen.

Beispiel: „Nennen Sie drei bekannte Programmiersprachen“

Die Antwort kann simpel über eine Liste sämtlicher bekannter Programmiersprachen abgeglichen und als richtig oder falsch bewertet werden.

Es können auch „algorithmische Aufgaben“ unter strukturierte Textaufgaben fallen, sofern nur das Ergebnis und nicht der Weg dorthin für die Korrektur von Belang ist. Möchte man beispielsweise nur das Ergebnis der Aufgabe „Nennen Sie den größten gemeinsamen

Teiler von 34 und 14“ muss die eingegebene Antwort „2“ lauten. Der Weg dahin ist irrelevant und es wird nur dieser Zahlenwert überprüft.

2.2.2 Unstrukturierte Textaufgabe / Freitext

Unter „Unstrukturierte Textaufgabe / Freitext“ werden alle Textaufgaben verstanden, die eine offene, in natürlicher Sprache zu beantwortende, Frage stellen. Die Musterlösung ist folglich ebenfalls in natürlicher Sprache vorgegeben.

Die automatische Vorkorrektur dieses Aufgabentyps ist direkt von der Aufgabenstellung der Computerlinguistik betroffen:

> *„Computerlinguistik erforscht die maschinelle Verarbeitung natürlicher Sprachen. Sie erarbeitet die theoretischen Grundlagen der Darstellung, Erkennung und Erzeugung gesprochener und geschriebener Sprache durch Maschinen.“* [3]

Das System soll in der Lage sein, die eingereichte Lösung gegen die Musterlösung abzugleichen. Hierbei müssen beide, in natürlicher Sprache geschriebenen, Texte, verarbeitet und miteinander verglichen werden. Die Herausforderung besteht darin ein geeignetes Konzept für diesen Schritt zu finden oder zu entwickeln.

2.2.3 Programmieraufgaben

Die letzte Kategorie bilden die Programmieraufgaben. Im Unterschied zu Freitext-Aufgaben werden Programmieraufgaben in einer Programmiersprache geschrieben und nicht in natürlicher Sprache verfasst. Allerdings tritt natürliche Sprache auch bei Programmieraufgaben auf (Beispiel: Ausgabe, Kommentare). Jedoch lassen sich bei Programmieraufgaben zunächst etliche Analysen erstellen. Bei kompilierenden Sprachen durchläuft der Code beispielsweise zunächst einige Analysephasen (Abbildung 2).

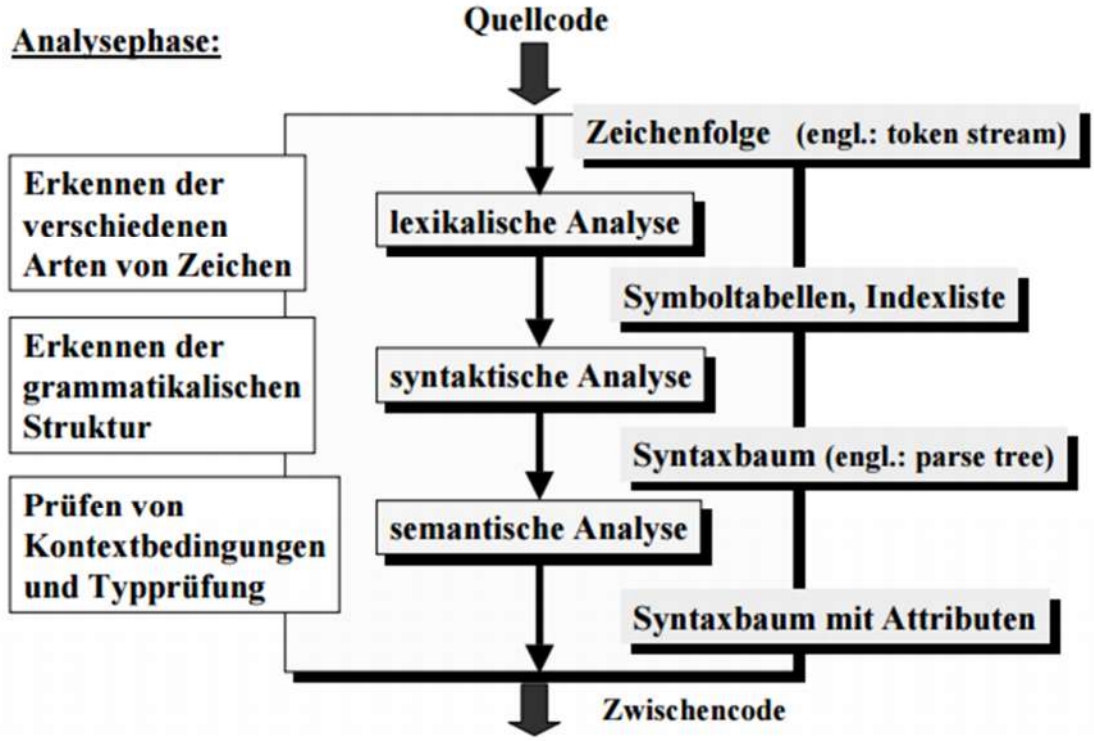


Abbildung 2: Analysephasen Compiler [4]

Des Weiteren können je nach Aufgabenstellung Testfälle erstellt und die Ausgabe auf Korrektheit überprüft werden. Soll ein Programm zu zwei gegeben Zahlen beispielsweise den größten gemeinsamen Teiler berechnen, lassen sich einige Testfälle erstellen, die Standard- und Grenz-/fehlerverursachende Fälle abdecken und die Ausgabe des Programms auf Korrektheit überprüfen.

2.3 Lernziele

Einhergehend mit dem Stellen von Übungsaufgaben kommt die Frage nach dem eigentlichen Lernziel auf. Innerhalb der Bloom'sche Taxanomie werden unterschiedliche Lernziele innerhalb des kognitiven Bereichs in sechs aufeinander aufbauende Hauptklassen eingeteilt: Wissen, Verstehen, Anwenden, Analyse, Synthese, Evaluation [5]. In [6] werden zwei Fragestellungen in Bezug auf die Operationalisierung von Lernzielen beschrieben. Unter anderem stellt sich die Frage was sich beim Lernenden bezüglich des Wissens und der Fertigkeiten verändern soll und wie man überprüfen kann, ob die Ziele erreicht wurden.

Gerade beim Lösen algorithmischer Aufgaben lassen sich gewisse Lernziele formulieren und überprüfen. Ein Algorithmus besteht wie in Abschnitt 2.1 erläutert aus eindeutigen Verarbeitungsschritten. Die Funktion des Algorithmus und die Anwendung der einzelnen Schritte sollte vom Lernenden verstanden und korrekt durchgeführt werden. Das System sollte demnach in der Lage sein die einzelnen Lösungsschritte überprüfen und vorkorrigieren zu können. Ziel ist also nicht nur die reine Prüfung des Endergebnisses, sondern jeden einzelnen (Rechen-)Schritt nachzuvollziehen. Durch diese Überprüfung kann dann eine Aussage getroffen werden, in wie weit die Funktion und Anwendung des Algorithmus verstanden und wie viele Rechen- oder Verständnisfehler dabei entstanden sind. Vergleicht man das mit der Bloom'schen Taxanomie erreicht man durch ein solches Vorgehen die dritte Stufe „Anwenden", da intensiv geprüft wird, ob der Algorithmus korrekt angewendet werden kann.

3 Related Work

In diesem Kapitel wird auf bereits existierende Systeme und wissenschaftliche Arbeiten bezüglich „Automatischer Vorkorrektur von Aufgaben“ eingegangen. Ein Überblick über die Methoden zur Textanalyse findet im letzten Abschnitt des Kapitels statt.

Es gibt zahlreiche Lösungsansätze und Systeme, die die automatische Vorkorrektur von Aufgaben unterstützen, sowie wissenschaftliche Beiträge über Möglichkeiten zur Automatisierung. Viele Hochschulen setzen Lernplattformen im Bereich des „E-Learnings“ ein. Günter Daniel Rey definiert den Begriff E-Learning als „das Lehren und Lernen mittels verschiedener elektronischer Medien“ [7]. Die Zielsetzung dieser Arbeit, ein Korrektursystem zu entwickeln, ist demnach im Bereich „E-Learning“ anzusiedeln. Web-basierende Lernplattformen, die über Autorentools zur Erstellung von Aufgaben verfügen, machen von E-Learning Gebrauch. Für diese Arbeit interessant waren die Möglichkeiten, die diese Lernplattformen bieten, um unterschiedliche Aufgabentypen zu korrigieren. Zunächst wollen wir uns einen Überblick über diese Systeme und ihre Eigenschaften verschaffen und feststellen, welche Aufgabentypen durch sie bereits gelöst werden können. Bei der Suche wurden nur kostenlose, frei-verfügbare Systeme berücksichtigt.

3.1 Lernplattformen

Die Internetseite CampusSource ist eine Initiative des Ministeriums für Schule, Wissenschaft und Forschung des Landes Nordrhein-Westfalen. Ziel der Initiative ist es unter anderem, die „Entwicklung von Softwaretechnologien, zur Unterstützung des Einsatzes neuer Medien in der Aus- und Weiterbildung“, in einen kooperativen Prozess zu fassen und durch die Bereitstellungen der verschiedenen Softwareangebote, die Weiterentwicklung existierender Software voranzutreiben [8]. Über die CampusSource-Börse wird Lernplattform-Software angeboten, die den Lizenzbestimmungen der General Public License (GPL) liegt. Für diese Arbeit ist das ein wichtiger Aspekt, da der Fokus auf kostenlose, Open Source Systeme gelegt wurde, um kostenlose Weiterentwicklung zu ermöglichen. Ein weiteres wichtiges Kriterium ist die Erweiterbarkeit via Plug-Ins. Die Lernplattform sollte über eine Plugin-Schnittstelle verfügen, um eigene Module einbinden zu können. Zwei Lernplattformen, ILIAS und Moodle sollen im Folgenden auf ihre Funktionalitäten im Hinblick auf automatische Vorkorrektur untersucht werden.

3.1.1 ILIAS

ILIAS (kurz für integriertes Lern-, Informations- und Arbeitskooperations-System) wurde ab 1997 an der Universität zu Köln entwickelt und wird mittlerweile weltweit eingesetzt. [9] Über die integrierte Plugin-Schnittstelle lässt sich ILIAS um selbstgeschriebene Plugins (beispielsweise in PHP oder Java geschrieben) erweitern.

3.1.2 Moodle

Moodle wurde ab 1999 von Martin Dougiamas entwickelt. Es ist ebenfalls durch Plugins erweiterbar und verfügt über einen sehr hohen Verbreitungsgrad. Aktuell nutzen laut der Internetseite „moodle.net“ über 53000 registrierte Websites mit über 69000 Benutzern in 225 verschiedenen Ländern Moodle. [10]

Tabelle 1 zeigt die unterstützten Aufgabentypen von ILIAS und Moodle in den angegebenen Versionen an.

Tabelle 1: Funktionsumfang ausgewählter E-Prüfungssysteme [11] [12] [13]

	ILIAS 4.4	Moodle 2.0
Anordnung (Reihenfolge)/ Zuordnung (Beziehung)	+/+	-/+
Single-/Multiple-Choice -Aufgabe	+	+
Lückentext-Aufgabe (Freitext, Auswahl)	+/+	+/+
Numerische Aufgabe (Zielbereich/Formel/Zufallswerte)	+/-/-	-/+/+
ImageMap-Aufgabe/Java-Applet-Aufgabe	+/+	-/-
Freitext	(+)	(+)
Legende: +: vorhanden/vorgesehen -: nicht vorhanden (+): indirekt vorhanden/herstellbar		

Beide Systeme beherrschen Zuordnungs-, Multiple-Choice- und Lückentext-Aufgaben. Bei ILIAS gibt es zudem die Möglichkeit numerische Aufgaben mit einem Zielbereich zu formulieren, während Moodle Aufgaben mit Formeln und Zufallswerten zulässt. Die Systeme sind in der Lage die Aufgaben auf Grundlage der vorgegebenen eindeutigen Antworten (Textaufgaben) bzw. Antwortbereiche (Multiple-Choice, numerischer Zielbereich) zu korrigieren. Freitext-Aufgaben lassen sich von keinem der beiden Systeme automatisch vorkorrigieren.

3.1.3 Zwischenfazit

Es zeigt sich, dass bereits einige Aufgabentypen durch die Funktionen der Lernplattformen automatisch vorkorrigiert werden können. Von den in Kapitel 2 definierten Aufgabentypen decken die vorgestellten Lernplattformen bereits Teile ab. Von den algorithmischen Aufgaben sind das einfache Formelberechnungen, die im Funktionsumfang der Lernplattform Moodle enthalten sind. Bei den Textaufgaben werden „strukturierte Textaufgaben“ bereits unterstützt. Einfache Abfragen nach Begriffen sind über Lückentexte oder Freitextfelder möglich. Nicht abgedeckt sind jene algorithmischen Aufgaben, die über das Anwenden einer einzelnen Formel in einem Rechenschritt hinausgehen. Das in Kapitel 2 erwähnte Aufgabenbeispiel Berechnung des größten gemeinsamen Teilers bleibt also offen. Ebenfalls nicht abgedeckt sind die Aufgaben die unter „unstrukturierte Textaufgaben / Freitext“ fallen. Im nächsten Abschnitt sollen die bisherigen wissenschaftlichen Beiträge zu Systemen automatischer Vorkorrektur näher untersucht werden.

3.2 Wissenschaftliche Beiträge

3.2.1 Automatische Korrektur von mathematischen Beweisen

In einer Doktorarbeit der Westfälischen Wilhelms-Universität Münster wurde mit dem Titel „Formatives E-Assessment in der Hochschullehre – Computerunterstützte Lernforschrittskontrollen im Informatikstudium“ das System „EASy“ entwickelt. Zielsetzung war „die Entwicklung eines E-Assessment-Systems, das zur Realisierung der gebotenen Potenziale in didaktischer, methodischer, organisatorischer und technischer Hinsicht beiträgt“. Unter anderem unterstützt es die Korrektur von Aufgaben mit mathematischen Beweisen. Das System verfügt über Benutzeroberflächen für Aufgabensteller und -Bearbeiter. Über das Verwenden verschiedener Beweisstrategien und Theoreme ist es möglich einen Beweis zu formulieren, den das System daraufhin auswertet und auf Korrektheit überprüft. Dafür wurde ein eigener Theorem-Beweiser entworfen. [14]

Eine automatische Korrektur von Aufgaben mit mathematischen Beweisen betrifft den Bereich „algorithmische Aufgaben“. Lassen sich mathematische Beweise automatisch vorkorrigieren, ist dies auch für das praktische Anwenden von Algorithmen möglich.

3.2.2 Automatische Korrektur von Programmieraufgaben

Zur automatischen Korrektur von Programmieraufgaben gibt es bereits etliche Systeme und Beiträge. Das an der Universität Passau entwickelte System „Praktomat“ wird beispielsweise von vielen Hochschulen eingesetzt. Mithilfe von Praktomat lassen sich eingesendete Programmieraufgaben automatisch testen und der Programmiercode auf Konventionen überprüfen. [15]

In einem Erfahrungsbericht der HTWK Leipzig [16] wurden Systeme zur automatischen Korrektur von Programmieraufgaben analysiert und verglichen. Unter anderem wurde für Aufgaben in der Programmiersprache Java hierbei eine Vorgehensweise vorgestellt. Bei der automatisierten Kontrolle eingesendeter Programmieraufgaben werden Funktionalitätstests durchgeführt. Die Funktionalitätstests bestehen aus in XML hinterlegten und von der verwendeten Programmiersprache unabhängigen Testfällen. Die Ergebnisse die durch das Anwenden der Testfälle entstehen werden mit SOLL-Werten abgeglichen. Zudem greifen eine „Styleguide-Überprüfung“ und eine „Plagiaterkennung“. [16]

In einer Ausarbeitung der Westfälischen Wilhelms-Universität Münster [17] wurde die Überprüfung von Code (hier Java-Code) in zwei Teile, die „statische Analyse“ und die „dynamische Analyse“ unterteilt. Unter der statischen Analyse wird die systematische Analyse des Quelltextes ohne Ausführung des Codes verstanden. Unter der dynamischen Analyse, wie in [15] und [16] die Ausführung des Codes mit konkreten Testfällen.

3.3 Computerlinguistik

In der Klasse der „Unstrukturierten Textaufgaben / Freitext“ bestehen die Abgaben aus natürlicher Sprache. Daher werden Methoden zur Verarbeitung benötigt. Ein Teilgebiet der Computerlinguistik ist die algorithmische Verarbeitung mittels eines Computers von natürlicher Sprache in Textform. [18] Um einen Text analysieren zu können wird häufig auf das Saarbrücker Pipelinemodell zurückgegriffen. Das Saarbrücker Pipelinemodell umfasst unter anderem folgende Schritte:

> *Tokenerkennung: Die Buchstabenkette wird in Bestandteile (Wörter, Sätze etc.) segmentiert.*

> *Morphologische Analyse: Personalformen oder Fallmarkierungen werden analysiert, um die grammatische Information zu extrahieren und die Wörter im Text auf Grundformen zurückzuführen. Dazu werden z.B. Lexika herangezogen.*
>
> *Syntaktische Analyse: Die Wörter jedes Satzes werden auf ihre strukturelle Funktion im Satz hin analysiert (z. B. Subjekt, Objekt, Modifikator, Artikel, etc.)*
>
> *Semantische Analyse: Den Sätzen bzw. ihren Teilen wird Bedeutung zugeordnet. Dieser Schritt umfasst potentiell eine Vielzahl verschiedener Einzelschritte, da Bedeutung schwer fassbar ist.*
>
> *Dialog- und Diskursanalyse: Die Beziehungen zwischen aufeinander folgenden Sätzen werden erkannt: Im Dialog könnte das z. B. Frage ↔ Antwort sein, im Diskurs beispielsweise eine Aussage und ihre Begründung, oder eine Aussage und ihre Einschränkung.* [19]

Um ein gutes Ergebnis bei der Verarbeitung von Freitext-Aufgaben erhalten zu können, werden im späteren Verlauf der Arbeit gewisse Schritte des Saarbrücker Pipelinemodells angewandt.

3.4 Fazit & Zielsetzung

Wir haben gesehen, dass Lernplattformen in der Lage sind etliche Aufgabentypen automatisch zu korrigieren. Unter anderem sind simple Textaufgaben, wie das Angeben eines einzelnen Begriffes und das Überprüfen auf Korrektheit, bereits automatisch korrigierbar. Bei Rechenaufgaben kann der Benutzer das Endergebnis eintragen und das System überprüft dieses auf Richtigkeit.

Das angesprochene System „EASy“ geht einen Schritt weiter und bietet unter anderem die Möglichkeit mathematische Beweise über Benutzeroberflächen zu formulieren und prüft diese automatisch auf Korrektheit.

Im Fachgebiet der Computerlinguistik werden Methoden zur Sprachverarbeitung untersucht und entwickelt.

In dieser Arbeit wurde die Wahl getroffen, ein System zu entwerfen, dass sich ausschließlich auf die Verarbeitung von Word-Dokumenten als Eingabemethode konzentriert. Es soll also ohne zusätzliche Benutzeroberfläche für die Bearbeiter der Aufgabe auskommen. Dabei soll es möglich sein sowohl algorithmische Aufgaben, als auch Freitext-Aufgaben im Word-Dokument zu bearbeiten und später im System vorkorrigieren zu lassen. Außerdem soll eingesendeter HTML Code vom System voranalysiert werden.

Der Entwicklungsansatz wird im folgenden Kapitel vorgestellt.

4 Eigener Ansatz

In diesem Kapitel wird der Weg zum Lösungsansatz, sowie der Lösungsansatz zur Problemstellung selbst, vorgestellt.

Wir haben gesehen, dass bereits einige Aufgabentypen durch existierende Systeme abgedeckt werden und es zahlreiche Beiträge verschiedener Institutionen zu Automatischen (Vor-)Korrekturen gibt. In dieser Arbeit wurde für die beiden Bereiche, „algorithmische Aufgaben“ und „Textaufgaben“ ein C#-Programm erstellt, das der korrigierenden Person die Möglichkeit bietet, Aufgaben und Musterlösungen zu erstellen. Das Programm korrigiert bearbeitete Aufgaben nach den von der korrigierenden Person vorgegebenen Lösungen, gibt eine Wertung ab und präsentiert das Ergebnis. Der Tutor kann daraufhin die Vorkorrektur kontrollieren, nachkorrigieren und Anmerkungen hinzufügen.

Für die in Kapitel 2 vorgestellte Klassifikation der Aufgaben wurden in dieser Arbeit separate Ansätze erstellt.

4.1 Wegfindung

Zunächst musste eine Grundlage für das System gefunden werden. Der erste, ursprüngliche Ansatz war, ein existierendes E-Learning System zu verwenden und das System zur Sprachverarbeitung und Bewertung darin einzubetten. Die Wahl fiel nach der Recherche hierbei zunächst auf Ilias. Wie in Abschnitt 3.1 bereits erläutert unterstützt Ilias etliche Aufgabentypen und bietet Lern-, sowie Kursmanagement. Über die Plugin-Schnittstelle kann man es um beliebige Komponenten erweitern und somit das System integrieren und bereits bestehende Funktionen nutzen.

Das Hauptproblem an dieser Idee war der Overhead und die erhöhte Komplexität, die durch die Implementation in Ilias erzeugt wird. Ilias benötigt in seiner aktuellen Version einen Apache Webserver (empfohlen 2.2.x oder höher), PHP/Datenbank Unterstützung, sowie die zusätzliche Software „ImageMick“ und „Info-Zip and Info-Unzip“. [20] Das System müsste in die von Ilias vorgegeben Strukturen integriert werden und mit der Datenbank kommunizieren, um die Punktevergabe bei automatischer Vor- und manueller Nachkorrektur abspeichern zu können. Zudem sollte das Einsendeformat beispielsweise ein Word-Dokument sein. Die unterstützten Aufgabentypen mit automatischer Korrektur sind bei Ilias in Form von Online-Eingaben gegeben, was im Widerspruch zu unserer Anforderung stand. Somit wurde diese Idee schlussendlich verworfen und ein anderer Lösungsansatz gesucht.

Wie in Kapitel 3 erläutert gibt es für Teilprobleme der Aufgabenstellung bereits gute Lösungsansätze. Diese sind allerdings sehr spezifisch, die Fragestellung nach einer automatischen Vorkorrektur von unseren vorgestellten algorithmischen und Freitext-Aufgaben bleibt ungelöst. Die finale Wahl fiel daher auf einen eigenen Ansatz unter Berücksichtigung existierender Kenntnisse und aktuellem Stand der Forschung. Es wurde eine Windows-Applikation programmiert, die das Erstellen und Einlesen von Word-Dokumenten ermöglicht und zur Vorkorrektur verschiedener Aufgabentypen verwendet werden kann.

4.2 Technologiekonzept

In diesem Abschnitt wird auf die verwendeten Technologien eingegangen.

Als Dateiformat der Einsendeaufgaben wurde vom Typ Word-Dokument ausgegangen. Als Programmiersprache zur Entwicklung des Systems wurde daher C# in Verwendung mit dem .NET Framework gewählt. Für die Entwicklungsumgebung wurde Visual Studio 2010 verwendet. Die graphische Benutzeroberfläche wurde mittels Windows Forms erstellt. Diese Kombination erwies sich als zielführend um ein Programm zu entwickeln, das Word-Dokumente ansprechen, erstellen oder bearbeiten muss. Als Dateiformat für die Hinterlegung der Aufgabe- und Korrekturinformationen wurde XML genutzt.

C# und das .NET-Framework

Das Programm wurde in der Programmiersprache C# geschrieben. Der Entwickler Microsoft hat 2000 C# als Teil des .NET Frameworks erstmals auf den Markt gebracht und entwickelt Sprache sowie Framework seitdem kontinuierlich. C# lehnt sich von der Syntax her stark an die von anderen C-Sprachen (C, C++) beziehungsweise Java an und ist ähnlich leistungsstark, wie andere aktuelle Sprachen. [21] Große Vorteile gegenüber anderen Sprachen sind die besonders gute Eignung für Windows-Applikationen sowie die Unterstützung im Zusammenspiel mit anderen Microsoft Produkten, wie Microsoft Word. Die in dieser Arbeit programmierte Windows Applikation verwendet C# in der Version 4.0.

Das .NET Framework ist die Microsoft-eigene Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. Das Framework besteht aus einer Laufzeitumgebung, verschiedenen Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen (Abbildung 3). [22]

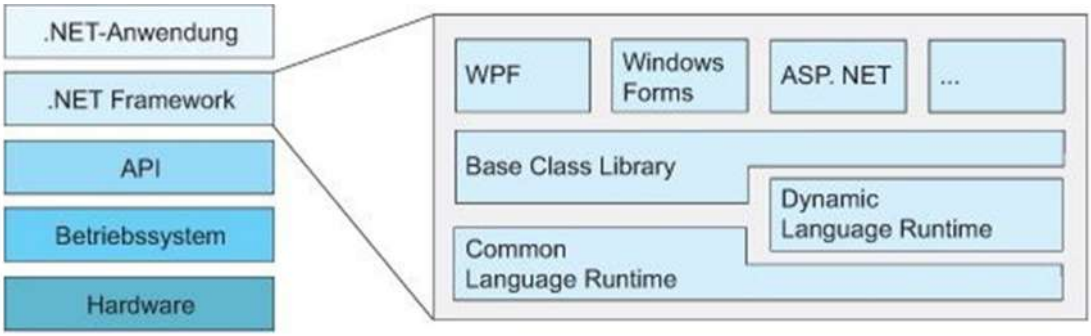


Abbildung 3: Blockdiagramm des .NET-Framework [22]

In dieser Arbeit wurde das .NET-Framework in der Version 4.0 verwendet.

Microsoft Visual Studio

Visual Studio ist die in das .NET-Framework integrierte Entwicklungsumgebung. Sie unterstützt neben C# eine Vielzahl weiterer Sprachen. Mithilfe von Visual Studio lassen sich neben Windows-Applikationen auch Websites, sowie Webservices oder Add-Ins für Microsoft Produkte entwickeln. [23]

Windows Forms

Windows Forms ist die verwendete Programmierschnittstelle (API), die in dieser Arbeit zur Erstellung der grafischen Benutzeroberfläche (GUI) verwendet wurde. Sie bietet Zugriff auf zahlreiche Steuerelemente und eignet sich sehr gut zum Erstellen grafischer Benutzeroberflächen von Windows Applikationen. [24]

Microsoft Word

Microsoft Word ist ein Textverarbeitungsprogramm vom Entwickler Microsoft. In dieser Arbeit wurde als Format für die Abgaben Word festgelegt. Das .NET-Framework bietet Methoden Word-Dokumente zu erstellen, zu bearbeiten und einzulesen.

XML

XML (kurz für: Extensible Markup Language) ist die Sprache, die zum Festhalten der Daten, die durch Aufgabenerstellung und Korrektur der Abgaben entstehen, verwendet wurde.

4.3 Grundlegendes Konzept und Eingabeformat

Die Frage die sich zunächst stellt ist, wie das Eingabeformat der Bearbeitung einer Aufgabe auszusehen hat und wie man dem System mitteilt, nach welchen Regeln es Zwischen- und Endergebnis zu überprüfen hat. Die eingesendete Aufgabe muss eine gewisse Struktur haben, die für das System greifbar ist. Um diese Struktur für jeden Aufgabentyp zu erreichen, verfügt das Programm über ein Auswahlmenü an Aufgabentypen, sowie einen Editor, in dem man zunächst die Aufgabe erstellt und die Musterlösung vorgibt. In den folgenden Unterkapiteln wird näher erläutert, wie dabei für jeden Aufgabentyp vorgegangen wird.

Nach Erstellen der Aufgabe legt das Programm selbst das Dokument in dem die Aufgabe bearbeitet wird in Form einer Word-Datei an. Innerhalb dieser Datei gibt es vorgegebene Bereiche, in welchen der Student die Aufgabe mit allen nötigen Schritten lösen kann. Das Dokument wird daraufhin dem Tutor zurückgeschickt. Dieser liest die Einsendeaufgaben in das Programm ein, woraufhin die automatische Vorkorrektur stattfindet. Abbildung 4 illustriert den Workflow.

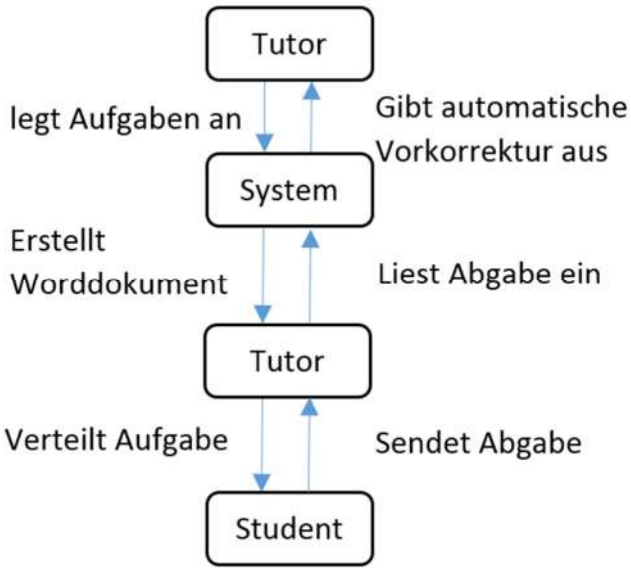


Abbildung 4: Workflow

4.4 Erstellung und Lösung algorithmischer Aufgaben

Algorithmische Aufgaben werden vom Bearbeiter der Aufgaben nach fest vorgegeben Schritten gelöst. Das System soll also nicht nur das Endergebnis auf Korrektheit zu überprüfen, sondern nachvollziehen können, ob der Student die einzelnen Schritte korrekt vollzogen hat und etwaige Folgefehler möglichst erkennen und visualisieren.

Zunächst war ein Editor angedacht in dem sich verschiedene algorithmische Aufgabentypen zur automatischen Vorkorrektur erstellen lassen. Das System sollte in der Lage sein, durch die im Editor erstellten Rechenschritte, eine Aufgabe zu korrigieren. Für die Berechnung des größten gemeinsamen Teilers würde man beispielsweise zunächst einen Rechenschritt im Editor definieren, bei dem die Zahlen mit Rest dividiert werden. Dann einen Rechenschritt in dem Dividend und Rest nun als Divisor und Dividend verwendet werden und schlussendlich, dass diese Rechenschritte solange wiederholt werden sollten, bis der Rest „0“ ergibt. Über einen solchen Editor könnten dann auch Aufgaben, wie das Umrechnen in andere Zahlensysteme oder weitere algorithmische Aufgaben erstellt werden. Jedoch zeigten sich bei dem Anlauf zur Umsetzung eines solchen Editors zu große Schwierigkeiten solche Rechenschritte mit entsprechenden Zusammenhängen zueinander definieren zu können.

Als realisierbareren und für die Zukunft erweiterbareren Ansatz wurde in dieser Arbeit, auf Grundlage der Formatierungsvorgaben des Systems, entschieden, den jeweiligen Algorithmus selbst zu programmieren und festzulegen nach welchen Vorgaben korrigiert werden soll. Als Beispiele wurden die Berechnung des größten gemeinsamen Teilers (ggT) nach Euklid und die Zahlenkonversion von Dezimal nach Binär genommen. Die in Kapitel 2 vorgestellten Schritte zur Berechnung des ggT werden zunächst vom Programm simuliert. Daraufhin wird das Word-Dokument mit einem vorgefertigten Formular angelegt. In dieses Formular trägt der Student jeden Rechenschritt ein und schickt das bearbeitete Dokument dem Tutor zurück. Beim Einlesen simuliert das Programm nun jeden Rechenschritt und vergleicht das Zwischenergebnis der eingesendeten Aufgabe mit dem zu erwartenden Ergebnis. Bei Ungleichheit wird das Zwischenergebnis als Rechenfehler angezeigt und ein vom Tutor definierter Punktabzug vorgenommen. Das Programm verwendet für jeden weiteren Rechenschritt, die vom Student verwendeten Zahlen samt etwaiger Rechenfehler und simuliert die Rechnung mit diesen Zahlen. Korrekte Ergebnisse werden als „korrekt“ - Folgefehler, also korrekte Rechnungen nach einem fehlerbehafteten Rechenschritt, werden als „korrekt / ff“ angezeigt und es werden keine weiteren Punkte abgezogen. Zuletzt wird außerdem das Endergebnis selbst mit dem korrekten Endergebnis verglichen und der Vergleich dem Tutor angezeigt.

Sind alle Rechenschritte korrekt ausgeführt worden und das Endergebnis ebenfalls korrekt, wird die volle Punktzahl für diese Aufgabe vom System angezeigt. Bei Fehlern wird die sich aus dem Abzug ergebende Punktzahl eingetragen. Der Tutor kann die Punktzahl bestätigen oder selbst eingreifen und eine Punktzahl vergeben. Daraufhin wird sie in das Worddokument an die vorgesehene Stelle geschrieben und das Dokument kann dem Studenten bei Bedarf zurückgesendet werden.

4.5 Erstellung und Lösung von Textaufgaben

Für die Erstellung und Lösung von Freitext-Aufgaben wurden aktuelle Sprachverarbeitungsmethoden angewandt um einen Abgleich zwischen Abgabe und Musterlösung zu schaffen. Für die im Bereich der Informatik spezifischen Textaufgaben als relevant be-

funden wurde in dieser Arbeit eine Verarbeitung sowie Klassifikation mittels der „Information Retrieval“-Methoden „Tokenisierung“, „Stoppwörter entfernen“, „Stemming“ sowie ein Schlüsselwörter Abgleich.

Die Abgabe wird zunächst eingelesen und dem Korrekter im Programm zusammen mit der Musterlösung visualisiert. Der Tutor kann selbst im Hinblick auf die Musterlösung entscheiden, welche Verfahren er zum Abgleich zwischen Abgabe und Musterlösung anwenden möchte.

Er hat dabei die im Folgenden vorgestellten Möglichkeiten, welche die einzelnen Schritte, die zur Auswertung des Systems führen, aufzeigen.

4.5.1 Verwendete Methoden

Tokenisierung

Grundlage des Vergleichs bildet die Tokenisierung.

> *„Vor der eigentlichen Analyse und Verarbeitung in elektronischer Form vorliegender Texte segmentiert man Dokumente in linguistische Einheiten, wie z.B. Wörter […] Ein solches Segmentierungsverfahren, das jedes Wort eines Textes erfasst, nennt man Tokenisierung.“* [18]

Aus Abgabe und Musterlösung wird mittels Tokenisierung die jeweilige Wortmenge (Tokenmenge) abgeleitet, die später für den Vergleich herangezogen wird.

Stoppwörter entfernen

Stoppwörter sind Wörter in einem Dokument, die weder Wissen noch inhaltliche Relevanz tragen. Sie nehmen keinen Einfluss auf die Rückgewinnung von Informationen. [25]

Um zu vermeiden, dass diese häufig auftretende Wörter ohne wesentliche Relevanz für den Inhalt und die Auswertung in das Ergebnis mit einfließen, besteht die Möglichkeit, diese zu erkennen und vor dem Abgleich zu entfernen. Sie fließen somit nicht in das Ergebnis der Auswertung mit ein. Für das Programm wurde eine Liste von 996 Stoppwörtern verwendet.

Stemming

Unter Stemming versteht man die Reduzierung eines Terms in verschiedenen morphologischen Varianten auf einen gemeinsamen Kern. Dieser Kern muss nicht unbedingt in der Sprache existieren. [26] Dadurch lassen sich bessere Ergebnisse bei der Ermittlung von Übereinstimmung zweier Texte erreichen.

Als Stemming-Algorithmus für das in dieser Arbeit erstellte System wurde eine deutsche Implementation des Porter-Stemmer-Algorithmus, verwendet. Der Porter-Stemmer Algorithmus wendet mehrere Reduktionsregeln an, bis eine gewisse Silbenzahl erreicht ist. [26]

Schlüsselwortabgleich

Dem Tutor wird die Möglichkeit gegeben, Schlüsselwörter der Musterlösung zu selektieren, die in der Abgabe vorkommen sollen. Er wählt aus einer Liste sämtlicher Begriffe der Musterlösung die Schlüsselwörter aus, woraufhin das Programm ermittelt, ob sie in der Abgabe vorkommen. Auch beim Schlüsselwortabgleich wird, sofern selektiert, die Methode „Stemming“ mittels Porter-Stemmer-Algorithmus angewandt.

Rechtschreibkorrektur

Schreibfehler sind für die Korrektur von zweierlei Bedeutung. Zunächst werden ohne Rechtschreibkorrektur falsch geschriebene Wörter bei einem Abgleich mit den Wörtern der Musterlösung nicht erkannt, sollten sie (in korrekt geschriebener Form) in der Musterlösung enthalten sein. Die Verfahren „Stopp-Wörter entfernen“, „Stemming“ und „Schlüsselwortabgleich“ funktionieren nur, wenn das Wort korrekt geschrieben wurde, da bei falscher Schreibweise nicht das jeweilige Stopp-Wort oder Schlüsselwort nicht gefunden, bzw. der Wortstamm ermittelt werden kann. Außerdem können Schreibfehler bei der Ermittlung der Punktevergabe helfen. Treten verhältnismäßig viele Schreibfehler auf, kann von einem schlechteren Ergebnis ausgegangen werden. Darum werden sie, sofern von der korrigierenden Person erwünscht, mit in die Auswertung der automatischen Vorkorrektur miteinbezogen.

Um ein möglichst gutes Ergebnis zu erhalten wurde die Microsoft Word interne Rechtschreibkorrektur verwendet. Das Programm kommuniziert mit der Word-Applikation und lässt sich Rechtschreibfehler, sowie die naheliegendste Korrektur zurückgeben. Für die Berechnung der Übereinstimmung mit der Musterlösung werden die korrigierten Wörter verwendet. Der Tutor kann jedoch entscheiden, ob er die Quote an richtig geschriebenen Wörtern in das Gesamtergebnis mit einfließen lassen möchte (siehe „Auswertung“).

4.5.2 Auswertung

Die schlussendliche Auswertung erfolgt über das Zusammentragen sämtlicher Informationen, die durch die gewählten Methoden, erhalten wurden. Zunächst durchläuft die Abgabe die Rechtschreibkorrektur. Danach werden Abgabe und Musterlösung tokenisiert. Es wird daraufhin ermittelt, wie viele Wörter der Abgabe mit der Musterlösung übereinstimmen und dem Tutor als absoluter und prozentualer Wert angezeigt. Sofern vom Tutor selektiert, werden die Stoppwörter vor dem Abgleich aus Abgabe und Musterlösung entfernt. Falls selektiert, wird zudem Stemming auf die Wörter der Abgabe und Musterlösung angewandt und ermittelt, wie viele Wortstämme übereinstimmen. Die Übereinstimmung wird ebenfalls prozentual und absolut angezeigt. Im letzten Schritt werden, sofern ausgewählt, sämtliche Schlüsselwörter im Abgabetext ermittelt und die Übereinstimmung dem Tutor visualisiert.

Aus den ermittelten Prozentwerten wird ein Gesamt-Prozentsatz ermittelt, der dem Tutor die ungefähre Übereinstimmung zwischen Abgabe und Musterlösung darstellen soll. Dieser wird wie folgt berechnet:

$$P_{Ges_1} = 0.7 * P_{TM} + 0.3 * P_{AM}$$
$$P_{Ges2} = 0.7 * P_{TMS} + 0.3 * P_{AM}$$
$$P_{Ges3} = 0.6 * P_{TMS} + 0.3 * P_{AM} + 0.1 * P_{FA}$$

P_{TM} … $Verhältnis\ gleiche\ Wörter\ zwischen\ Abgabe\ und\ Musterlösung\ in\ Prozent$

P_{TMS} … $Verhältnis\ gleiche\ Wörter\ zwischen\ Abgabe\ und\ Musterlösung\ mit\ Stemming\ in\ Prozent$

P_{AM} … $Verhätnis\ gleiche\ Wörter\ und\ gesamte\ Wortzahl\ der\ Abgabe\ in\ Prozent$

P_{FA} … $Anteil\ der\ falsch\ geschriebenen\ Wörter\ innerhalb\ der\ Abgabe\ in\ Prozent$

Formel 1: Berechnung der Prozentsätze ohne Schlüsselwörter

Mit Schlüsselwörtern:

$$P^*_{Ges_1} = 0.5 * P_{Ges_1} + 0.5 * P_{SW}$$
$$P^*_{Ges_2} = 0.5 * P_{Ges_2} + 0.5 * P_{SW}$$
$$P^*_{Ges_3} = 0.5 * P_{Ges_3} + 0.5 * P_{SW}$$

P_{SW} …$Verhältnis\ getroffene\ Wörter\ Abgabe\ zu\ Schlüsselwörter\ in\ Prozent$

Formel 2: Berechnung der Prozentsätze mit Schlüsselwörter

Auf Basis dieses Prozentsatzes wird ein Bewertungsvorschlag in Abhängigkeit der zu erreichenden Maximalpunktzahl erstellt. Der Tutor kann dann selbst entscheiden, in wie weit er auf den Bewertungsvorschlag eingeht und vergibt die erreichten Punkte selbst. Die letzten Endes selbstständige – statt automatisierte – Vergabe der Punkte durch den Tutor ist ein wichtiger Schritt. Gesetzlich dürfen zum aktuellen Stand Prüfungsleistungen ausschließlich „von Person bewertet werden, die selbst mindestens die durch die Prüfung festzustellende oder eine gleichwertige Qualifikation besitzen." [27] Da es sich bei den Übungsaufgaben einer Universität um Prüfungsleistungen handeln kann, wurde deshalb dieser Ansatz gewählt. Das System erstellt also auf Grundlage der erfassten Daten nur eine Empfehlung, die Bewertung selbst nimmt stets der Tutor vor.

4.6 Lösung von Code Aufgaben

In dem in dieser Arbeit entworfenen System werden sich die gewonnenen Erkenntnisse aus Kapitel 3 zu Nutze gemacht. Das System assistiert den Tutor bei der Korrektur von Code Aufgaben durch Validierung des Codes. In unserem Test-Set beschränkten wir uns auf HTML-Code, der über den kostenlosen WDG HTML Validator von „Web Design Group" validiert wird. Der Validation Service ist seit 1999 im Einsatz und wurde über die Jahre hinweg den neuen HTML-Versionen angepasst [28]. Das System schickt den abgegeben Code an den Validation Service, dieser wertet ihn aus und das System empfängt die Rückmeldung, die in verarbeiteter Form dem Tutor präsentiert wird (Abbildung 5).

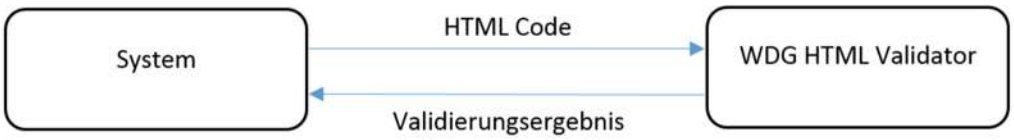


Abbildung 5: Schematische Darstellung HTML Validierung

Auf Basis der Auswertung kann nun die manuelle Korrektur vorgenommen und die Punktzahl vom Tutor vergeben werden. Fehlerfrei validierter Code kann für den Tutor ein Anzeichen einer richtigen Lösung sein.

4.7 Festhalten der Daten

Das System leitet für jede erstellte Aufgabe sowohl ein dazugehöriges Word-Dokument mit vorgegebenen Feldern zur Verteilung und Bearbeitung, als auch die zugehörigen Informationen der Aufgabe, wie Aufgabentyp, Punktzahl, gegebenenfalls automatischer Punktabzug und weiterer Informationen, aus. Um diese Informationen festhalten zu können wird eine zweite Datei zu jeder Aufgabe im Format „XML“ erstellt (Abbildung 6).

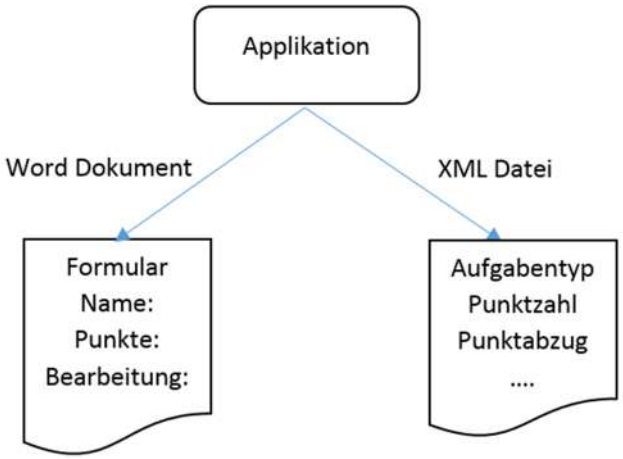


Abbildung 6: Schematische Darstellung der Aufgabenausleitung

5 Realisierung des Ansatz

Im Folgenden Kapitel wird die konkrete Realisierung des in Kapitel 4 erläuterten Ansatzes vorgestellt.

5.1 Schritte zur Aufgabenerstellung & Korrektur

5.1.1 Erstellen einer Aufgabe

Jede Aufgabenerstellung läuft nach dem gleichen Muster ab. Der erste Schritt für den Tutor ist das Erstellen einer Aufgabe im Programm. Dazu wählt er zunächst aus einer Liste (siehe Abbildung 7), um welche Art von Aufgabe es sich handelt. Das Programm verfügt über die Typen „Textaufgabe (frei)“, „Code“ – und die algorithmischen Aufgabentypen „GGT“ und „Zahlenkonvertierung“.

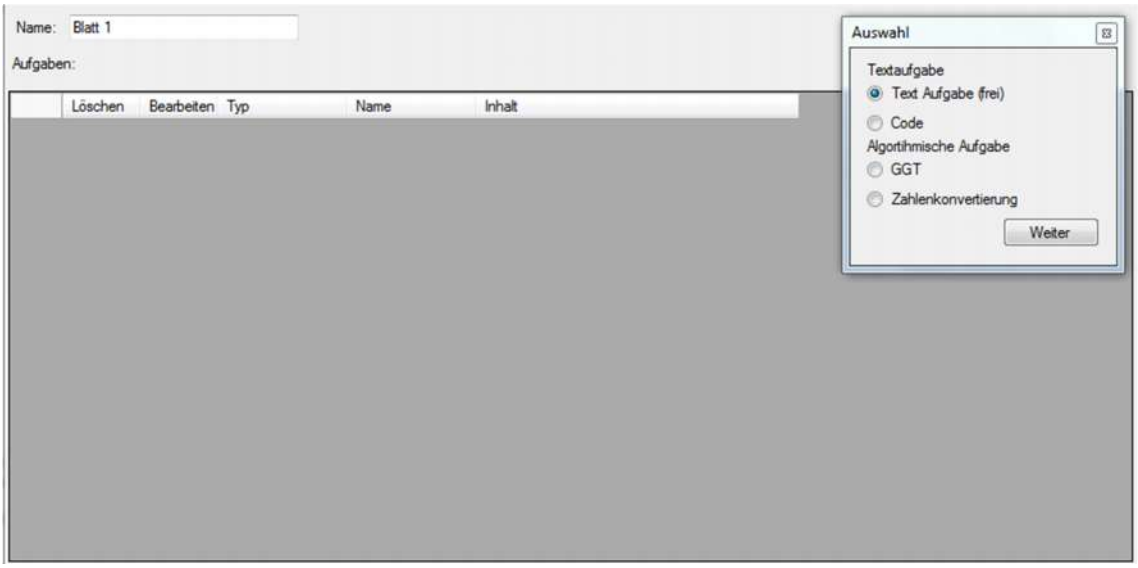


Abbildung 7: Screenshot vom Auswahlmenü beim Erstellen einer Aufgabe

Im folgenden Schritt legt der Tutor die Aufgabenparameter fest. Bei jeder Art von Aufgabe sind Aufgabentext und Punktzahl anzugeben. Bei freien Textaufgaben kommen außerdem die Musterlösung (siehe Abbildung 8) und bei Code Aufgaben die zu verwendende Sprache (in dieser Arbeit auf HTML beschränkt) hinzu.

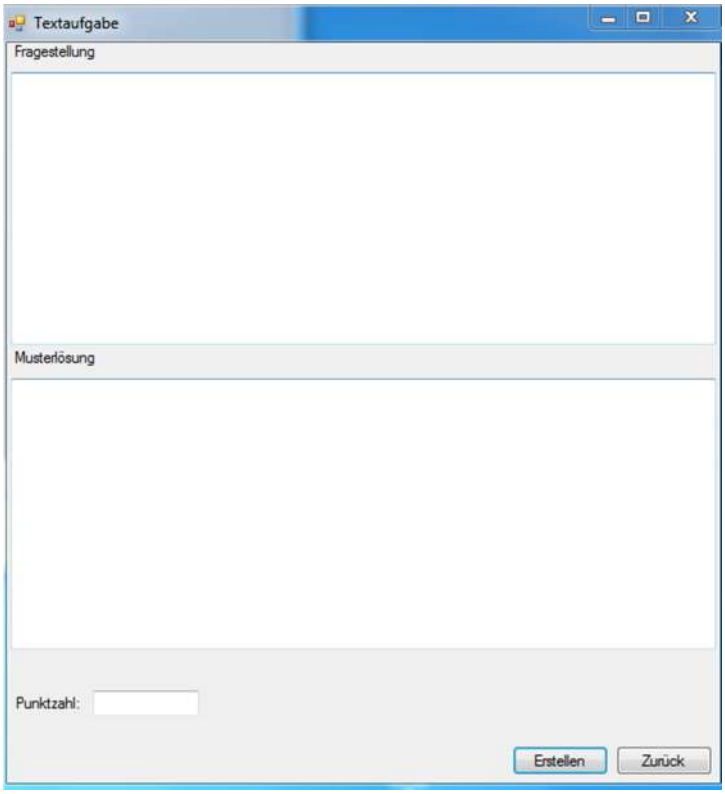


Abbildung 8: Darstellung Eingabe Textaufgabe

Bei algorithmischen Aufgaben (siehe Abbildung 9) kommen die zu verwendenden Zahlen (GGT) bzw. die zu konvertierende Zahl und die beiden Zahlensystem, in dieser Arbeit auf Dezimal zu Binär beschränkt, (Zahlenkonversion) hinzu. Außerdem wird der Punktabzug bei einem Rechenfehler, sowie bei einem Übertragfehler, im Voraus vom Tutor festgelegt.

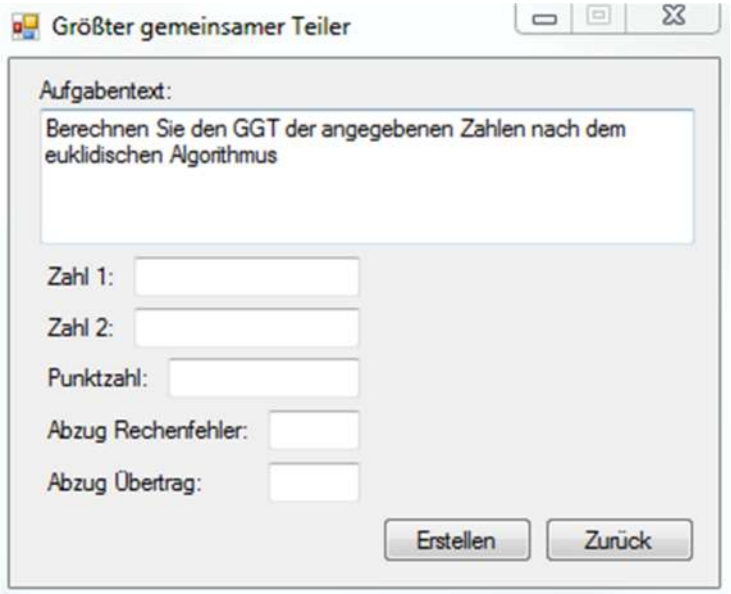


Abbildung 9: Darstellung Eingabe Größter Gemeinsamer Teiler

Nach Festlegen der Parameter erstellt das Programm das Worddokument und die XML-Datei der Aufgabe. Der Formularkopf des Worddokuments ist immer fest vorgegeben und besteht aus einem Feld für den Namen der bearbeitenden Person, einem für die Punktzahl, die später bei der Korrektur automatisch eingetragen wird und einem für die Fragestellung (Abbildung 10).

Name:
Erreichte Punkte:
Aufgabe [Nummer]   ***Frage:***

Abbildung 10: Formularkopf Word-Dokument

Für die beiden algorithmischen Aufgabentypen dieser Arbeit wird das Formular um eine Tabelle erweitert, in die der Student die einzelnen zur Berechnung des Ergebnisses notwendigen Zahlenwerte einträgt. Das Programm ermittelt die benötigten Rechenschritte beim Erstellen der Aufgabe und legt entsprechend viele Zeilen an. Es wird berücksichtigt, dass bei eventuellen Rechenfehlern, mehr Rechenschritte benötigt werden, als bei korrekter Durchführung. Deshalb werden bei jeder Aufgabe mehr Zeilen, als eigentlich benötigt, erstellt. Abbildung 11 zeigt den Ausschnitt der Tabelle eines, vom Programm erstellten, Word-Dokuments zur Berechnung des größten gemeinsamen Teilers der Zahlen 1728 und 258. Nach euklidischem Algorithmus sind fünf Rechenschritte nötig, um die Aufgabe zu lösen. Angelegt wurden zusätzlich noch zwei weitere Zeilen. Verrechnet sich die bearbeitende Person, kann sie also dennoch ihre Rechnung innerhalb der Tabelle zu Ende führen.

Aufgabe 1 ***Frage: Berechnen Sie den GGT der angegebenen Zahlen nach dem euklidischen Algorithmus***			
1728 und 258			
GGT:			

Abbildung 11: Tabellenformular des Word-Dokuments

In der letzten Zeile des Formulars wird ein Feld zur Angabe des Endergebnisses angelegt.

5.1.2 Erstellen einer Korrektur

Zur Erstellung einer Korrektur wählt der Tutor die bei der Aufgabenerstellung erzeugte XML-Datei aus. Das Programm verarbeitet die darin erhaltenen Parameter der Aufgabe. Der Aufgabentyp gibt dem Programm vor, inwiefern Abgabe und Vorkorrektur visualisiert werden. Abbildung 12 zeigt die angelegte XML-Datei der gleichen Aufgabe wie in vorherigem Abschnitt, dem Berechnen des größten gemeinsamen Teilers der Zahlen 1728 und 258.

```
<?xml version="1.0"?>
<Sheet name="Blatt 1">
    <Exercise name="Aufgabe 1" type="EX_ALG_GGT">
        <Question>Berechnen Sie den GGT der angegebenen Zahlen nach dem euklidischen Algorithmus</Question>
        <Zahl1>1728</Zahl1>
        <Zahl2>258</Zahl2>
        <Punktzahl>5</Punktzahl>
        <AbzugRechenfehler>0,5</AbzugRechenfehler>
        <AbzugUebertrag>3</AbzugUebertrag>
    </Exercise>
</Sheet>
```

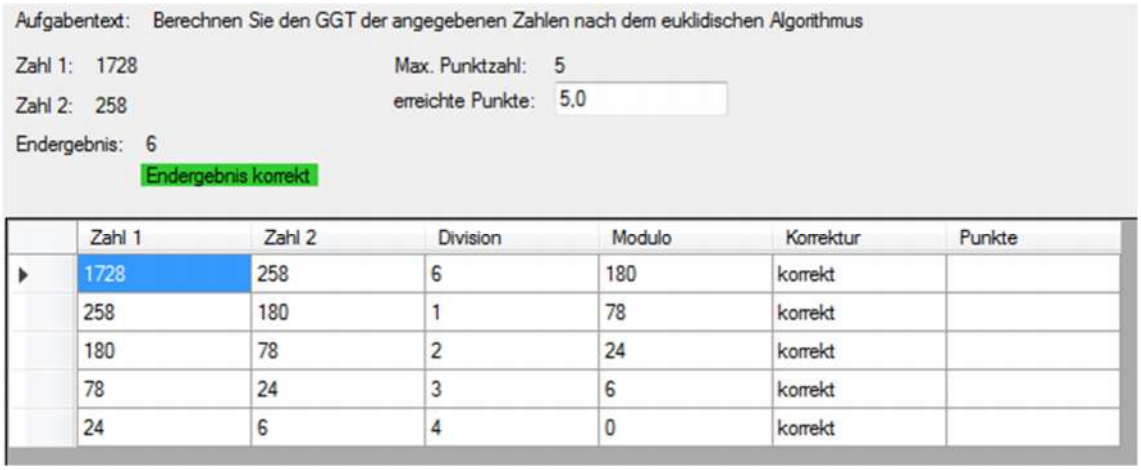
Abbildung 12: Ausgeleitete XML Datei

Die Visualisierung wird im nächsten Abschnitt erläutert.

5.2 Visualisierung der Vorkorrektur

5.2.1 Visualisierung der algorithmischen Aufgabentypen

Um Abgaben algorithmischer Aufgabentypen möglichst übersichtlich und verständlich dem Tutor anzuzeigen, wird nach dem Einlesen das bearbeitete Word-Formular im Programm in einem tabellarischen Steuerelement dargestellt. Eine zusätzliche Spalte „Korrektur“ gibt in jeder Zeile an, ob die Rechnungen korrekt durchgeführt oder Fehler gemacht wurden. Eine weitere Spalte Punkte stellt den daraus folgenden Punktabzug dar, der sich aufgrund des zuvor festgelegten Punktabzugs pro Rechenschritt ergibt. Abbildung 13 zeigt die Anwendung des euklidischen Algorithmus zur Berechnung des GGTs der Zahlen 1728 und 258 komplett richtig durchgeführt.



Zahl 1	Zahl 2	Division	Modulo	Korrektur	Punkte
1728	258	6	180	korrekt	
258	180	1	78	korrekt	
180	78	2	24	korrekt	
78	24	3	6	korrekt	
24	6	4	0	korrekt	

Abbildung 13: Visualisierung Vorkorrektur einer korrekt gelösten Abgabe

Da in Abbildung 13 alle Rechenschritte korrekt durchgeführt wurden, trägt das Programm automatisch die volle Punktzahl ein und gibt dem Tutor den Kommentar „Ergebnis korrekt“ zurück.

In Abbildung 14 hingegeben werden die Fehler angezeigt und die Punkte abgezogen. Die entstandenen Fehler werden pro Zeile summiert und mit dem festgelegten Punktabzug multipliziert. Die Summe wird von der maximal erreichbaren Punktzahl abgezogen und vorgeschlagen. Ab dem ersten Rechenfehler simuliert das Programm alle weiteren Rechenschritte mit den (falschen) Zahlenwerten der Abgabe. Sind die folgenden Schritte korrekt berechnet worden wird dem Tutor dies mit der Bemerkung „korrekt / ff“ (Folgefehler) angezeigt und es findet kein weiterer Punktabzug statt.

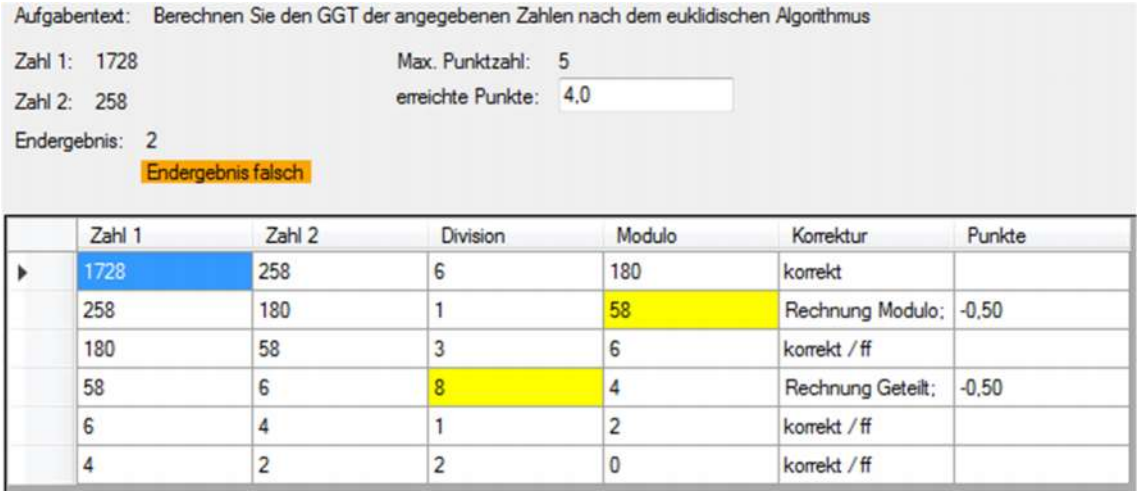


Abbildung 14: Visualisierung Vorkorrektur einer fehlerbehafteten Abgabe

5.2.2 Visualisierung der Freitext-Aufgaben

Für das Korrigieren von Freitext-Aufgaben werden die Abgabe und die Musterlösung in einer gemeinsamen Oberfläche nebeneinander dargestellt, damit für den Tutor ein möglichst schneller Abgleich zwischen Abgabe und Musterlösung stattfinden kann. Der Tutor wählt über Checkboxen welche Methoden er für die automatische Vorkorrektur anwenden möchte. Nach Bestätigung der Auswahl ermittelt das System die in Kapitel 4.5 erläuterten einzelnen Werte und berechnet die Prozentsätze sowie den, für die Korrektur relevanten, Gesamtprozentsatz. Die Ergebnisse werden dem Tutor tabellarisch dargestellt (Abbildung 15)

Abbildung 15: Tabellarische Darstellung der Auswertung

Zur erweiterten Hilfestellung für den Tutor lässt sich über die Benutzeroberfläche die ermittelte Übereinstellung für jeden Schritt grafisch nachvollziehen. Die gefundenen Übereinstimmungen für die einzelnen Methoden (ohne bzw. mit Stemming, Schlüsselwortabgleich) in Abgabetext und Musterlösung werden in verschiedenen Farben markiert (Abbildung 16).

Abbildung 16: Farbliche Darstellung der Auswertung

Die farblichen Markierungen sollen dem Tutor das Ermitteln einer richtigen Punktevergabe erleichtern und assistieren beim Nachvollziehen des zustande gekommen Ergebnisses der automatischen Vorkorrektur.

Eine Erweiterung dieser Darstellung ist die entworfene Hybriddarstellung, die das Gegenüberstellen zweier Abgaben ermöglicht. Die Benutzeroberfläche erweitert sich um einen dritten Teil, in dem eine zusätzliche Abgabe dargestellt ist. Beide Abgaben werden vom System nach den ausgewählten Methoden des Tutors korrigiert (Abbildung 17).

Abbildung 17: Hybriddarstellung

Der Tutor kann sämtliche durch die Vorkorrektur ermittelten Werte in der Hybrid-Ansicht gegenüberstellen. Dies kann sehr hilfreich sein, wenn der Tutor die Ergebnisse der Vorkorrektur nachvollziehen und mit Ergebnissen anderer Korrekturen vergleichen möchte. Es soll ihm außerdem helfen, seine eigene Punktevergabe für mehrere Abgaben leichter zu treffen. Hat er eine Abgabe bereits bewertet, kann er daraus eventuell direkt die Punktzahl für die zweite Abgabe ableiten, oder, falls noch keine der beiden Abgaben bewertet wurde, sie zugleich in einem Schritt bewerten.

5.2.3 Visualisierung der Programmieraufgaben in HTML

Um HTML Aufgaben korrigieren zu können wählt der Tutor zunächst die abgegebene HTML-Datei aus und startet die Validierung. Nachdem das System die Eingabe validiert hat, erscheint das Ergebnis der Validierung in der Benutzeroberfläche. Auf Wunsch kann er sich zusätzlich den Code in einem zweiten Fenster anzeigen lassen. Der Tutor kann nun bereits auf Basis der Validierung Entscheidungen bezüglich der Korrektur der Aufgabe treffen und wahlweise in der Nachkorrektur den Code im Einzelnen durchgehen. Abbildung 18 illustriert die Gegenüberstellung zwischen Validierungsergebnis und eingegebenen Code anhand eines simplen Beispiels mit einigen Zeilen fehlerhaftem HTML-Code.

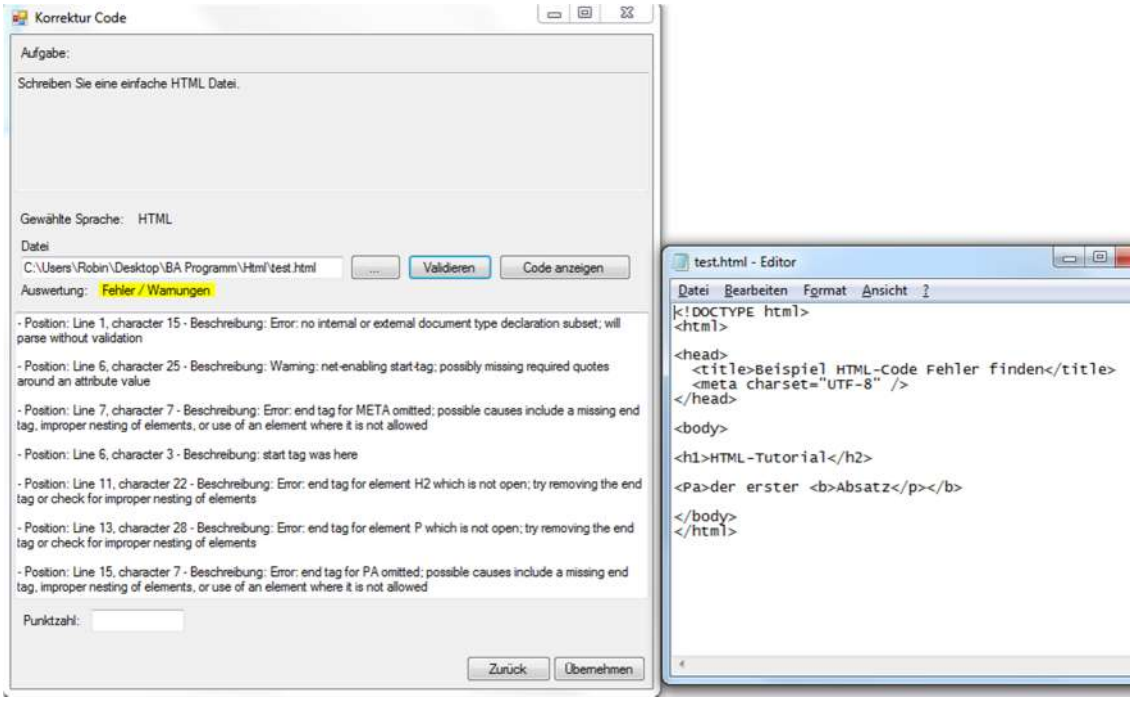


Abbildung 18: Visualisierung Korrektur Programmieraufgabe

5.3 Auswertung und Nachkorrektur

Nach der automatischen Vorkorrektur erfolgt die Auswertung und Nachkorrektur beziehungsweise schlussendliche Punktevergabe durch den Tutor. Es ist wichtig anzumerken, dass die ermittelte Punktzahl nur eine Empfehlung des Programms ist. Die Bewertung der Aufgabe erfolgt durch die Entscheidung des Tutors, die Empfehlung zu verwenden oder zu verwerfen.

Bei algorithmischen Aufgaben kann der Tutor zunächst die Korrekturbemerkungen des Programms durchgehen und sich dann entscheiden, inwieweit er die ermittelte Punktzahl übernimmt oder gegebenenfalls nachjustiert.

Bei Textaufgaben geschieht die automatische Vorkorrektur nicht über einen festgelegten Abzug pro falschem Rechenschritt, sondern über den Gesamt-Prozentsatz, der sich aus den einzelnen Prozentsätzen wie in Abschnitt 4.5.2 erläutert, zusammensetzt und die maximale Punktzahl ermittelt. Tabelle 2 zeigt die Aufteilung in prozentuale Bereiche.

Tabelle 2: Bewertungsbereiche

Gesamtbewertung Vorkorrektur (%)	Faktor Punktevergabe im Verhältnis zur Gesamtpunktzahl
70-100	1,0
0-70	Bewertung Vorkorrektur / 100%

Die Aufteilung in diese Bereiche erfolgte unter der Annahme, dass eine Übereinstimmung natürlicher Sprache von 70 Prozent oder mehr – auch nach Anwenden der Methoden „Stoppwörter entfernen“ und „Stemming“ – als korrekt gelöste Aufgabe verstanden werden kann, während der Bereich unter 70% in direkter Abhängigkeit zur prozentualen Übereinstimmung steht. Zukünftig werden die Zuhilfenahme empirischer Ermittlungen und maschineller Lernverfahren hierbei sicherlich helfen können, genauere Bewertungen finden zu können.

Schlussendlich werden die Punkte in das Word-Dokument eingetragen und können der jeweiligen Person, die die Aufgabe bearbeitet hat, zurückgesandt werden (Abbildung 19).

Abbildung 19: Ausgeleitete Punktzahl in das bearbeitete Word-Dokument

Zudem wird eine XML-Datei erstellt in der ein Verweis auf Abgabe-Datei und vergebene Punktzahl gespeichert wird. Im Hinblick auf zukünftige Weiterentwicklungen unter Zuhilfenahme empirischer Ermittlungen oder maschineller Lernverfahren können diese Dateien zur (Massen-)Verarbeitung verwendet werden.

5.4 Klassenkonzept & Erweiterbarkeit

Das Programm wurde wie in Kapitel 4.2 erläutert in C# geschrieben. Wie in Abschnitt 4.2 erläutert, gehört C# mit steigender Tendenz zu den weit verbreiteten Programmiersprachen. Der Umgang ist vergleichsweise leicht. Somit sind gute Grundvoraussetzungen gegeben, das Programm zukünftig zu erweitern.

Das Grundlayout der vorgestellten Aufgabentypen ist durch das erstellte Programm in Form von Word-Formularen vorgegeben und kann zukünftig für die Implementierung weiterer Ansätze verwendet und mit geringem Aufwand erweitert werden. Für das Erstellen und Lösen von Aufgaben innerhalb des Systems wurde zunächst eine für alle Aufgabenarten gültige Basisklasse definiert. Für „algorithmische Aufgaben“ und „unstrukturierte Textaufgaben“ wurden jeweils C#-Klassen geschrieben, die von der Basisklasse erben (Abbildung 20). Eine Enumerator-Klasse beinhaltet die verschiedenen Aufgabenarten.

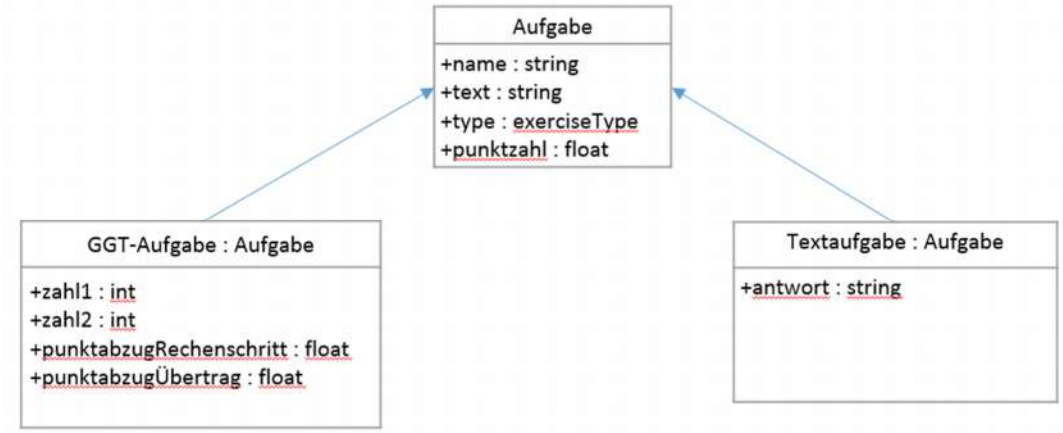


Abbildung 20: UML-Klassendiagramm

Zukünftig können weitere Aufgabetypen integriert werden, indem die Enumerator-Klasse entsprechend erweitert und eine eigene Klasse für den Aufgabentyp geschrieben wird. Die Strukturen der Basisklasse und des Word-Layouts können hierbei wiederverwendet werden.

Die XML-Dateien, die vom Programm erstellt werden, können bei der Auswertung von (Massen-)Daten im Bereich des maschinellen Lernens verwendet werden. In diesen Dateien werden Informationen zur Aufgabe und zur Korrektur festgehalten.

Die Inanspruchnahme externer Internet-Services wurde in dieser Arbeit innerhalb der Unterkategorie „Programmieraufgaben“, durch die Verwendung des HTML-Validator-Services von Web Design Group realisiert. Über die Integration bzw. Verwendung diverser externer Services, Datenbanken oder Applikationen könnte zukünftig auch die Korrektur andere Aufgabentypen abgedeckt bzw. bestehender Aufgabentypen verbessert werden.

6 Analyse und Auswertung

Dieses Kapitel beschäftigt sich mit der Analyse des Programms und der Auswertung der Ergebnisse, die es liefert. Es werden Vor- und Nachteile zu den zwei Aufgabenfeldern „Algorithmische Aufgaben“ und „Textaufgaben“, sowie Möglichkeiten des Zusammenspiels mit maschinellen Lernverfahren aufgezeigt.

6.1 Algorithmische Aufgaben

Im Folgenden Abschnitt werden zunächst die Ergebnisse der automatischen Vorkorrektur von algorithmischen Aufgaben besprochen. Als Beispiel dient eine Aufgabe der Aufgabenblätter „Grundlagen der Informatik I“ - die Berechnung des größten gemeinsamen Teilers (ggT) nach Euklid mit den Zahlen 1728 und 258.

Wir betrachten die vom Programm ermittelten Ergebnisse anhand von drei Abgaben. Eine komplett korrekte Eingabe. Eine Abgabe mit Rechenfehlern in mehreren Zwischenschritten, sowie zuletzt eine Abgabe mit Übertragungsfehlern und falschem Endergebnis.

Abbildung 21 zeigt eine korrekte Bearbeitung der Aufgabe und die Darstellung der Vorkorrektur durch das Programm.

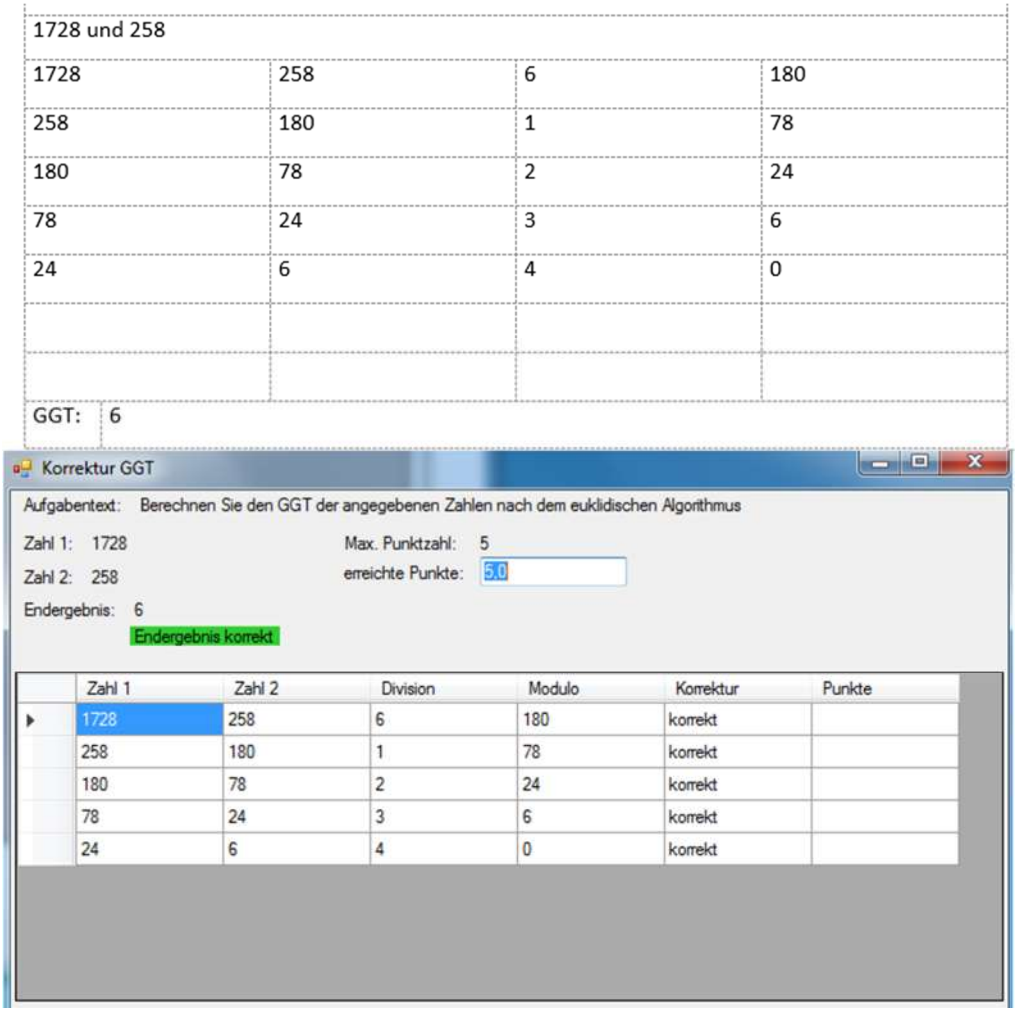


Abbildung 21: Darstellung einer korrekten Bearbeitung

Die erste vollkommen korrekte Abgabe entspricht der Musterlösung. Sie wird vom Programm eingelesen, jeder Rechenschritt als korrekt bewertet und die volle Punktzahl vorgeschlagen. Da jeder Rechenschritt vom Programm simuliert und für Korrekt befunden wird, bleibt die manuelle Korrekturarbeit hier erspart, die Schritte müssen nicht vom Tutor nachkontrolliert werden.

Abbildung 22 zeigt die zweite Abgabe mit einem Rechenfehler im zweiten und vierten Rechenschritt. Die restlichen Rechenschritte wurden korrekt ausgeführt.

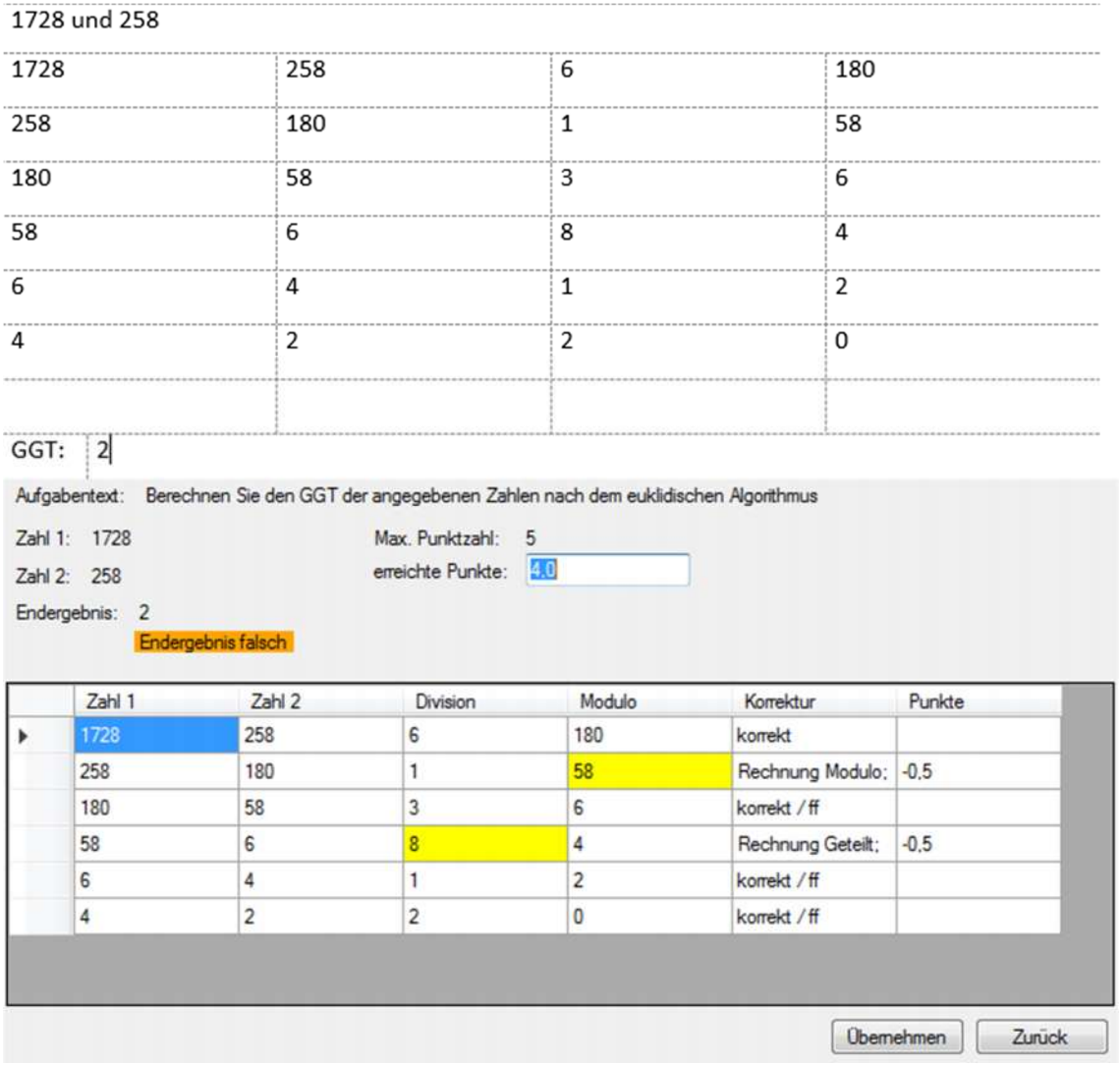


Abbildung 22: Darstellung einer fehlerhaften Bearbeitung (Rechenfehler)

Das Programm zeigt den Fehler und Bewertungsvorschlag an. Sofern vom Tutor erwünscht kann er den Vorschlag ohne weiteres übernehmen und erspart sich die Korrekturarbeit auch in diesem Fall. Daraus erschließt sich ein großer Vorteil bei Korrekturen von Aufgaben mit einer sehr hohen Zahl an Rechenschritten und/oder einer hohen Anzahl an zu korrigierenden Abgaben. Die sich ständig wiederholenden Rechenvorschriften manuell zu korrigieren stellt sich als ermüdend dar und verleitet zu Fehlern bei der Korrektur. Durch die Simulation innerhalb des Programms und automatischer Überprüfung jedes einzelnen Rechenschritts, ist es irrelevant wie viele Fehler an welcher Stelle einer Rechnung gemacht wurden. Die Fehler werden erkannt und dem Tutor direkt angezeigt. Es bleibt daraufhin ihm überlassen, ob er diese in die Bewertung mit einfließen lässt.

In Abbildung 23 werden eine Abgabe mit Übertragungsfehlern, sowie die Auswertung des Programms, gegenübergestellt.

1728 und 258			
1728	258	6	180
258	6	43	0
GGT:	6		

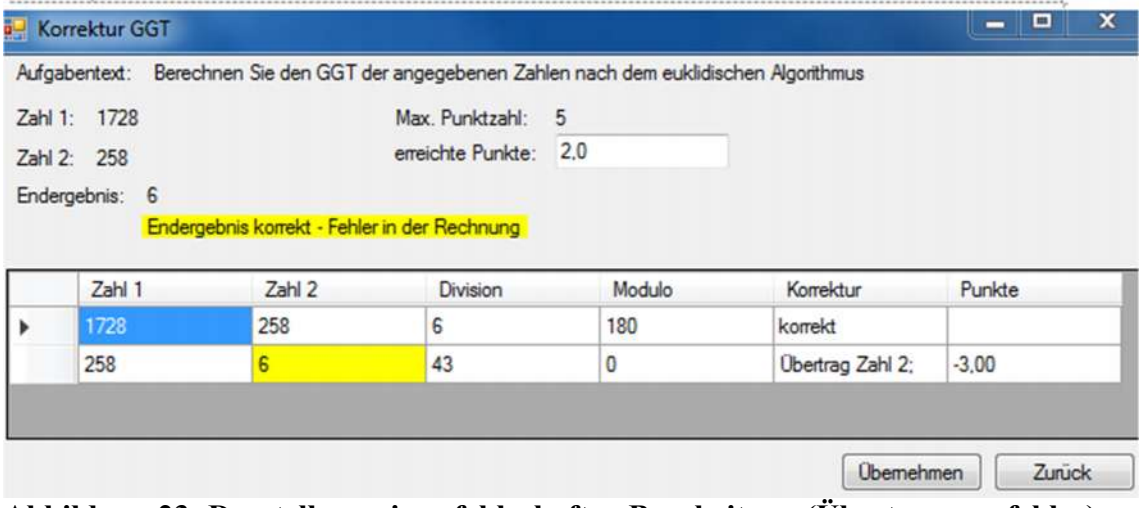


Abbildung 23: Darstellung einer fehlerhaften Bearbeitung (Übertragungsfehler)

Dieses Beispiel zeigt auch, dass kleine Fehler innerhalb einer Rechnung, die dennoch zum richtigen Endergebnis führen, erkannt werden können.

Die graphische Benutzeroberfläche zeigt für jeden Fehler die Begründung, sowie den aus dem Fehler resultierenden Punktabzug an. Möchte der Tutor pauschal jeden Rechen- und Übertragungsfehler mit dem festgelegten Abzug bewerten, kann er direkt das vorgeschlagene Ergebnis der automatischen Vorkorrektur verwenden. Er kann alternativ im Hinblick auf wiederkehrende Fehler, die durch das Programm anhand häufig vorkommender Kommentare (z.B. „Fehler: Division“) veranschaulicht werden, das Ergebnis aber auch beispielsweise verwerfen und selbst entscheiden, wie er die Fehler gewichtet.

6.1.1 Vorteile der automatischen Vorkorrektur bei algorithmischen Aufgaben

Bei algorithmischen Aufgaben mit fest vorgegebenen Rechenschritten, wie in vorherigem Kapitel verwendetem Beispiel, ist eine brauchbare Auswertung durch das Programm relativ gut zu erreichen. Fehler werden für jeden Schritt erkannt und ein Punktabzug pro Fehler kann vom Programm automatisch vorgenommen werden. Korrekt gelöste Aufgaben können automatisch mit voller Punktzahl versehen werden und der Tutor benötigt insbesondere für diese Aufgaben keine Korrekturzeit mehr. Das Ziel, das mühsame Korrigieren richtiger Abgaben dem Tutor abzunehmen ist für dieses Aufgabenfeld innerhalb der Struktur des Programms also erreicht. Jedoch wirft eben diese Struktur auch Nachteile auf, die im nächsten Abschnitt erläutert werden.

6.1.2 Nachteile der automatischen Vorkorrektur bei algorithmischen Aufgaben

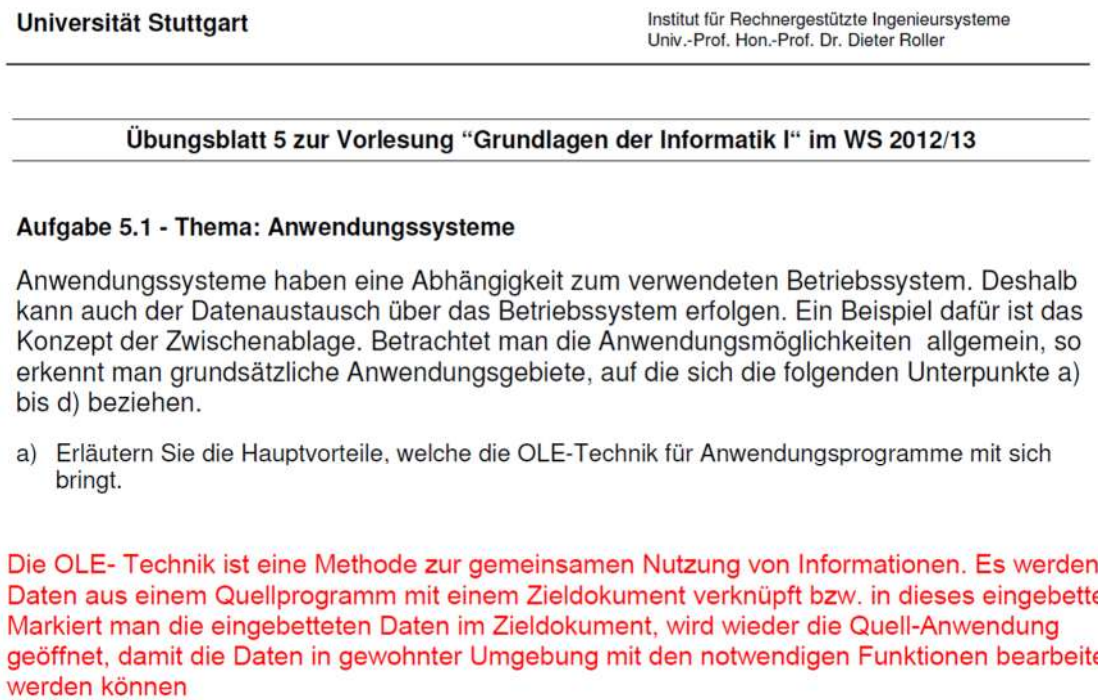
Das Programm erstellt wie in Kapitel 4 erläutert vorgegebene Formulare zur Bearbeitung von algorithmischen Aufgaben. Die Zahlenwerte, die bei jeder Rechnung entstehen, müssen in dieses Formular an richtiger Stelle eingetragen werden. Dies muss den bearbeitenden Personen zunächst mitgeteilt werden, was im Vorfeld einen zusätzlichen Mehraufwand verursacht. Zudem wird mit einer solchen Vorgabe eines Formulars, wie es in diesem Programm der Fall ist, ein Teil der Lösungsfindung der bearbeitenden Person vorweg genommen. Die feste Vorgabe einer Tabelle schließt sehr viele falsche Lösungsansätze im Vorfeld aus, die korrekte Lösungsfindung wird erleichtert. Verdeutlichen lässt sich dies beispielsweise bei der Berechnung des größten gemeinsamen Teilers durch Division mit Rest. Das vom Programm vorgegebene Formular erstellt für jeden benötigten Zahlenwert (Dividend, Divisor, Quotient, Rest) je eine Spalte. Somit ist klar, dass pro Rechenschritt nicht mehr und nicht weniger als vier Zahlenwerte vorkommen können / dürfen. Diese im Vorfeld stattfindende Einschränkung mag bei simplen algorithmischen Aufgaben wie bei unseren Beispielen nicht von allzu großer Bedeutung sein. Für komplexere Aufgabenstellungen, die eine stärkere Vorgabe des Eingabeformulars erfordern, könnte es jedoch im Konflikt zum Wunsch der eigenständigen Lösungsfindung stehen.

Weiterhin entstehen Probleme, sobald die Aufgabenstellung mehrere Wege zur Lösungsfindung offen lässt. Ist beispielsweise die Reihenfolge bestimmter Rechenschritte für die Richtigkeit des Ergebnisses irrelevant, muss ein anderer Ansatz gefunden werden. Dies könnte entweder durch das Implementieren weiterer / aller Lösungswege in das System sein oder, und eher wahrscheinlicher, durch das Verschieben solcher Aufgabenstellungen in den Bereich „Textaufgaben“.

6.2 Textaufgaben

6.2.1 Freitext-Aufgaben

In diesem Abschnitt werden die Korrekturergebnisse des Programms von Textaufgaben analysiert und ausgewertet. Als Beispiel dient eine Aufgabenstellung aus den Übungsblättern „Grundlagen der Informatik 1“. Für die Aufgabe analysieren wir die automatische Vorkorrektur zweier Abgaben und vergleichen die jeweiligen Unterschiede. Zunächst die Aufgabe samt Musterlösung. Abbildung 24 zeigt die Aufgabenstellung und die Musterlösung.



Universität Stuttgart

Institut für Rechnergestützte Ingenieursysteme
Univ.-Prof. Hon.-Prof. Dr. Dieter Roller

Übungsblatt 5 zur Vorlesung "Grundlagen der Informatik I" im WS 2012/13

Aufgabe 5.1 - Thema: Anwendungssysteme

Anwendungssysteme haben eine Abhängigkeit zum verwendeten Betriebssystem. Deshalb kann auch der Datenaustausch über das Betriebssystem erfolgen. Ein Beispiel dafür ist das Konzept der Zwischenablage. Betrachtet man die Anwendungsmöglichkeiten allgemein, so erkennt man grundsätzliche Anwendungsgebiete, auf die sich die folgenden Unterpunkte a) bis d) beziehen.

a) Erläutern Sie die Hauptvorteile, welche die OLE-Technik für Anwendungsprogramme mit sich bringt.

Die OLE- Technik ist eine Methode zur gemeinsamen Nutzung von Informationen. Es werden Daten aus einem Quellprogramm mit einem Zieldokument verknüpft bzw. in dieses eingebettet. Markiert man die eingebetteten Daten im Zieldokument, wird wieder die Quell-Anwendung geöffnet, damit die Daten in gewohnter Umgebung mit den notwendigen Funktionen bearbeitet werden können

Abbildung 24: Text Aufgabe samt Musterlösung

Nun die zwei verschiedenen Abgaben:

Abgabe 1:

„*OLE Objekte können miteinander verknüpft oder eingebettet werden. Beim Auswählen eines eingebetteten Objekts wird die ursprüngliche Anwendung geöffnet und die Daten können darin bearbeitet werden.*"

Abgabe 2:

> *„Object Linking and Embedding (OLE, engl. Objekt-Verknüpfung und -Einbettung) ist ein von Microsoft entwickeltes Objektsystem und Protokoll, das die Zusammenarbeit unterschiedlicher (OLE-fähiger) Applikationen und damit die Erstellung heterogener Verbunddokumente ermöglichen soll.*
>
> *Zum Beispiel kann eine Tabelle, welche mit einem Tabellenkalkulationsprogramm erstellt wurde, in ein Textdokument eingebunden werden. Die Besonderheit ist dabei, dass diese eingebettete Tabelle direkt aus dem Textprogramm heraus per Doppelklick mit dem ursprünglichen Programm bearbeitet werden kann. Dieses läuft dazu – erkennbar am geänderten Menüaufbau – als Rumpfprogramm innerhalb der Dokument-Applikation.*
>
> *OLE-Objekte können entweder verlinkt (Object Linking) oder eingebettet (Embedding) werden. Bei einer Verlinkung wird nur eine Referenz auf das eingebundene Objekt erstellt und im Dokument gespeichert, während bei einer Einbettung eine Kopie des Objekts im Dokument gespeichert wird. Diese Verlinkung bzw. Einbettung kann u. a. via Drag & Drop oder Copy & Paste erstellt werden. Der Vorteil des Einbettens besteht darin, dass das Verbunddokument von den Quelldateien der eingebundenen Objekte unab-*

hängig ist. Allerdings benötigen durch Embedding erzeugte Verbunddokumente auch mehr Speicherplatz als die durch Object Linking erzeugten.“ [29]

Während Abgabe 1 in etwa eine korrekte Antwort zur Aufgabe darstellt, wurde für Abgabe 2 ein Ausschnitt des ersten Google Resultats zum Suchbegriff „OLE“, der Wikipedia Artikel „Object Linking and Embedding“ verwendet. Zwar enthält dieser Ausschnitt einige Begriffe, die auch in der Musterlösung vorkommen, die Frage wird jedoch nicht präzise beantwortet, sondern einfach ein Text zum Oberbegriff „OLE“ aus dem Internet kopiert.

Tabelle 3 zeigt die Auswertung des Programms der beiden Abgaben spaltenweise, zunächst mit der Methode „Schlüsselwörter entfernen“ selektiert bzw. de-selektiert und ohne „Stemming“. Tabelle 4 zeigt die Auswertung mit „Stemming“. Zum Vergleich stehen in der letzten Spalte die Werte bei vollkommener Übereinstimmung zwischen Abgabe und Musterlösung (Abgabe entspricht Musterlösung). „Token getroffen“ gibt die Anzahl übereinstimmender Token zwischen Abgabe und Musterlösung an. Die Zeile „Verhältnis Treffer / Anzahl Token Musterlösung“ gibt die prozentuale Übereinstimmung zwischen den Token in der Abgabe und den Token in der Musterlösung an. Die Zeile „Verhältnis Treffer / Anzahl Token Abgabe“ gibt das Verhältnis zwischen den getroffenen Token und der Anzahl an Token der Abgabe an.

Tabelle 3: Ergebnis Vorkorrektur ohne Stemming und Schlüsselwörter

	Abgabe 1 ohne / mit Stopp-wort-Entfernen	Abgabe 2 ohne / mit Stopp-wort-Entfernen	Musterlösung ohne / mit Stopp-wort-Entfernen
Token getroffen	12 / 9	19 / 6	42 / 24
Anzahl Token Musterlösung	42 / 24	42 / 24	42 / 24
Verhältnis Treffer / Anzahl Token Musterlösung (%)	28,57 / 37,50	45,24 / 25,00	100 / 100
Verhältnis Treffer / Anzahl Token Abgabe (%)	54,55 / 60,00	15,83 / 8,11	100 / 100
Gesamtbewertung (%)	36,36 / 44,25	36,42 / 19,93	100 / 100

Deutlich zu erkennen sind zunächst die Unterschiede, die vom (Nicht-)Entfernen der Stoppwörter hervorgerufen werden. Die prozentuale Übereinstimmung zwischen Abgabe und Musterlösung steigt bei Abgabe 1 um knapp 8 Prozent. Das Verhältnis der übereinstimmenden Wörter zur Gesamtzahl an Wörtern in der Abgabe steigt ebenfalls, um über 5 Prozent. Bei Abgabe 2 sinkt hingegen die Übereinstimmung beim Anwenden von Stoppwörtern entfernen. Der deutlich längere Text enthält mehr Stoppwörter, die auch in der Musterlösung vorkommen, als Abgabe 1. Die Übereinstimmung ohne das Entfernen von Stoppwörtern suggeriert somit eine, verglichen mit Abgabe 2, bessere Lösung. Das

Anwenden der Methode Stoppwörter entfernen korrigiert das Ergebnis jedoch, die prozentuale Übereinstimmung fällt von 45,24 auf 25,00. Das Verhältnis der getroffenen Token zur Gesamtanzahl der Token der Abgabe sticht als weiterer wichtiger Faktor bei der Bewertung heraus. Der Text von Abgabe 2 ist ungleich länger als der von Abgabe 1. Somit ist die Chance Token der Musterlösung zu treffen ebenfalls größer. Das Verhältnis der Treffer zur Abgabe selbst korrigiert diesen Wert. So ist die Übereinstimmung zur Musterlösung ohne Stoppwörter Entfernen bei Abgabe 2 mit 45,24 Prozent zunächst höher als die von Abgabe 1 mit 28,57 Prozent. Das Verhältnis zur Tokenanzahl der Abgabe 2 ist mit 15,83 Prozent aber drastisch niedriger als das bei Abgabe 1 mit 54,55 Prozent. Das spiegelt sich in der Gesamtbewertung dann wieder, die bei beiden Abgaben in etwa gleich ist. Einzelne Werte in einem sehr niedrigen Bereich sind für den Tutor also ein Indikator für eine nicht präzise, möglicherweise nicht zutreffende Lösung. Insbesondere das Verhältnis getroffener Token zur Gesamtanzahl innerhalb der Abgabe kann zeigen, ob einfach ein langer, zum Thema passender aber allgemein gehaltener Text kopiert / verfasst wurde – oder ob es sich um eine (kurze), eventuell präzisere Antwort zur Fragestellung handelt.

Tabelle 4: Ergebnis Vorkorrektur mit Stemming ohne Schlüsselwörter

	Abgabe 1 ohne / mit Stopp-wort-Entfernen	Abgabe 2 ohne / mit Stopp-wort-Entfernen	Musterlösung ohne / mit Stopp-wort-Entfernen
Token getroffen	12 / 8	19 / 6	40 / 23
Token Musterlösung	40 / 23	40 / 23	40 / 23
Verhältnis Treffer / Anzahl Token Musterlösung (%)	30,00 / 34,78	47,50 / 26,09	100 / 100
Verhältnis Treffer / Anzahl Token Abgabe (%)	60,00 / 61,54	17,76 / 9,23	100 / 100
Gesamtbewertung (%)	39,00 / 42,81	38,58 / 21,03	100 / 100

Inklusive Anwenden von Stemming auf Abgabe und Musterlösung zeigt sich erneut ein Anstieg der Prozentwerte beim Entfernen von Stoppwörtern. Der Unterschied ist nicht mehr so groß, wie zuvor ohne Stemming, dennoch ist bei Abgabe 1 ein Anstieg der Übereinstimmung von fast fünf Prozent und beim Verhältnis zur Tokenanzahl der Abgabe von etwas über 1,5 Prozent zu sehen. Im Vergleich zu den Werten ohne die Anwendung von Stemming sind – ohne Entfernen der Stoppwörter – die Prozentwerte allesamt leicht angestiegen (Übereinstimmung zur Musterlösung: 1,43 Prozent; Verhältnis zur Abgabe: 4,45 Prozent; Gesamtbewertung 2,64 Prozent), was der Vermutung entspricht, dass bei Reduktion auf einen gemeinsamen Kern, eine höhere Übereinstimmung vorzufinden ist. Interessanterweise trifft das nicht auf die sämtliche Werte zu, die wir beim Anwenden von Stemming und Stoppwörter entfernen erhalten haben. Hier ist zwar das Verhältnis übereinstimmender Token zur Gesamtanzahl an Token der Abgabe um 1,54% gestiegen, Die prozentuale Übereinstimmung zur Musterlösung ist jedoch um 2,72% gesunken, was

zur Folge hat, dass die Gesamtbewertung um 1,44% gesunken ist. Grund hierfür kann das mehrmalige Verwenden eines Begriffes sein, der auf denselben Kern zurückgeführt wird. Das Anwenden von Stemming sorgt bei Abgabe 2 für ähnliche Veränderungen des Ergebnisses, wie bei Abgabe 1. Hier taucht das Phänomen einer sinkenden Gesamtbewertung jedoch nicht auf, sämtliche Prozentwerte und somit auch das Gesamtergebnis steigen leicht an.

Tabelle 5 und Tabelle 6 zeigen die Auswertung mit der zusätzlichen Option „Schlüsselwörter markieren“. Als Schlüsselwörter definiert wurden „Ole“, „Daten“, „Quellprogramm“ und „verknüpft“. Abgabe 1 hat drei von vier Schlüsselwörter, während in Abgabe 2 nur ein Schlüsselwort getroffen wurde.

Tabelle 5: Ergebnis Vorkorrektur ohne Stemming mit Schlüsselwörtern

	Abgabe 1 ohne / mit Stopp-wort-Entfernen	Abgabe 2 ohne / mit Stopp-wort-Entfernen	Musterlösung ohne / mit Stopp-wort-Entfernen
Bewertung ohne Stemming (%)	36,36 / 44,25	36,42 / 19,93	100 / 100
Schlüsselwörter getroffen	3	1	4
Schlüsselwörter insgesamt	4	4	4
Verhältnis (%)	75,00	25,00	100
Gesamtbewertung (%)	55,68 / 59,63	30,71 / 22,47	100 / 100

Tabelle 6: Ergebnis Vorkorrektur mit Stemming mit Schlüsselwörtern

	Abgabe 1 ohne / mit Stopp-wort-Entfernen	Abgabe 2 ohne / mit Stopp-wort-Entfernen	Musterlösung ohne / mit Stopp-wort-Entfernen
Bewertung mit Stemming (%)	39,00 / 42,81	38,58 / 21,03	100 / 100
Schlüsselwörter getroffen	3	1	4
Schlüsselwörter insgesamt	4	4	4
Verhältnis (%)	75,00	25,00	100
Gesamtbewertung (%)	57,00 / 58,90	31,79 / 23,02	100 / 100

Die Ergebnisse zeigen, dass Abgabe 1 in der Gesamtwertung somit erneut gestiegen ist. Abgabe 1 setzt sich somit im Vergleich zu Abgabe 2 als mögliche korrekte Lösung durch.

Im nächsten Abschnitt werden die Vorteile, die sich durch die automatische Korrektur von Freitext-Aufgaben ergeben, aufgezeigt.

6.2.2 Vorteile der automatischen Vorkorrektur von Freitext-Aufgaben

Die durch das vom Programm ermittelten Werte der Übereinstimmung zwischen Musterlösung und Abgabe einer Freitext-Aufgabe können den Tutor bei der Korrektur unterstützen. Bevor er selbst mit der Korrektur beginnt, verschafft das Programm einen Überblick der Übereinstimmung der beiden Texte mithilfe der erläuterten Methoden. So kann zum Beispiel schneller entschieden werden, ob eine Antwort mitunter korrekt sein könnte, ohne sie sich davor durchzulesen. Die verschiedenen angegebenen Prozentual- und Absolutwerte lassen sich zu dem mithilfe der in Abschnitt 5.2.2 erläuterten Hybridansicht gegenüberstellen. Dies stellt einen zusätzlichen Vorteil für den Tutor dar um schneller und einfacher zu einem Korrekturergebnis zu kommen, da er Abgabe, Vorkorrektur und seine eigene Punktevergabe einer anderen Abgabe und deren Vorkorrektur gegenüberstellen kann. Erreicht eine Abgabe eine hohe prozentuale Übereinstimmung kann der Tutor bereits mit einer gewissen Wahrscheinlichkeit von einer korrekten Antwort ausgehen. Schlussendlich ist also schnelleres und effizienteres Korrigieren möglich. Es hängt jedoch stark von einigen Faktoren ab, wie brauchbar das Ergebnis der Vorkorrektur wirklich ist. Außerdem muss beachtet werden, dass bei Freitext die letztendliche Bewertung durch den Tutor und nicht durch die automatische Vorkorrektur des Systems erfolgen muss. Die Nachteile der automatischen Vorkorrektur werden im nächsten Abschnitt erläutert.

6.2.3 Nachteile der automatischen Vorkorrektur von Textaufgaben

Wie in Abschnitt 4.5 und 6.2.1 erläutert erfolgt die Auswertung über verschiedene Übereinstimmungswerte zwischen Abgabe und Musterlösung unter Zuhilfenahme mehrerer Verfahren. Eine solche Vorkorrektur kann gute Ergebnisse liefern, diese sind jedoch sehr stark von den gewählten Begriffen in Abgabetext und Musterlösung anhängig. Werden sinngemäß richtige Begriffe verwendet, die jedoch nicht in der Musterlösung vorkommen, wird das System keine Übereinstimmung finden und eine dementsprechend niedrige Bewertung ausgeben. Ein möglicher Lösungsansatz dieses Problems wäre die Zuhilfenahme eines Synonymabgleichs. Somit würden Wörter gleicher Bedeutung als Treffer markiert und im Ergebnis der automatischen Vorkorrektur repräsentiert werden.

Ein weiterer Nachteil ist der Aufwand der für den Tutor beim Auswerten der Vorkorrektur entsteht. Zunächst muss er entscheiden, welche Methoden er für den Vergleich heranzieht. Darüber hinaus muss er, sofern von ihm erwünscht, die Schlüsselwörter, die im Text vorkommen sollen erst aussuchen und dann in der Benutzeroberfläche selektieren. Die verschiedenen prozentualen Ergebnisse, sowie das Gesamtergebnis der Vorkorrektur müssen vom Tutor gegengehalten und eingeschätzt werden, da sie für ihn nur eine Hilfestellung sind. Er muss selbst entscheiden wieviel Bedeutung er den Werten im Einzelnen zuspricht und inwiefern diese Werte ausschlaggebend für seine Korrektur sind.

Ausschlaggebend ist zudem die Musterlösung selbst. Ist die Musterlösung im Hinblick auf eine automatische Vorkorrektur ungünstig formuliert, hat das erheblich negative Auswirkungen auf das Ergebnis der Vorkorrektur. Wird bei der Musterlösung beispielsweise eine Wortwahl getroffen, die im allgemeinen Sprachgebrauch nur selten vorkommt, wird eine viel geringere Übereinstimmung zu Abgabetexten vorliegen. Abhilfe könnte hier ein Anpassen der Musterlösung für eine automatische Vorkorrektur oder das Verwenden und Abgleichen mehrerer Musterlösungen schaffen.

Abschließend bleibt die Frage nach der Zeitersparnis. Bei langen Texten ist eine Auswertung über die Vorkorrektur sicherlich hilfreich, um insbesondere korrekte Lösungen erkennen zu können, ohne den gesamten Abgabetext durchlesen zu müssen. Dennoch gilt auch hier, dass der Tutor die eigentliche Punktevergabe erteilt und für die Korrektur verantwortlich ist. Es liegt also in seinem Ermessen, inwiefern die vom System vorgeschlagenen Werte für die Punktevergabe eine Rolle spielen. Bei kurzen Abgabetexten hingegen die nur relativ wenig Korrekturaufwand erfordern ist die Frage, ob es nicht schneller ist, sie von Hand zu korrigieren, statt sie in das System einzulesen, die unterschiedlichen Methoden zur Korrektur auszuwählen und die Ergebnisse der Vorkorrektur einzuschätzen.

In Kapitel 7 werden auf mögliche Lösungen dieser Probleme und mögliche künftige Erkenntnisse nochmals im Detail eingegangen.

6.2.4 HTML Aufgaben

Das Programm überprüft ob der abgegebene HTML Code gültig ist. Eine inhaltliche Prüfung findet nicht statt. Die Überprüfung über den WGD HTML Validator brachte sehr gute Ergebnisse bei der Überprüfung von erstellten Testdaten mit fehlerhaftem Code hervor. So wurden alle Fehler im Code erkannt und ausgegeben. Die Angabe der Position im Code und die Gegenüberstellung des abgegebenen Codes, erleichtern das schnelle Nachvollziehen der Erläuterung des Fehlers. Der Tutor kann folglich Gebrauch von der Auswertung des Systems machen, muss aber dennoch selbst überprüfen, ob die Aufgabe korrekt gelöst wurde. Die Vorkorrektur ist dabei nur eine Hilfestellung. Die Angaben zu etwaigen Fehlern im Code mit Zeile und Position können dem Tutor helfen, Fehler des Studenten schnell und einfach nachvollziehen zu können und stellen somit eine Erleichterung bei der Bewertung der Aufgabenbearbeitung dar.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde der Versuch getätigt ein System zur automatischen Vorkorrektur von Aufgaben zu entwickeln. Dabei wurde zu einem Rundumschlag ausgeholt, Lösungsansätze für verschiedenartige Aufgabentypen, wie algorithmische Aufgaben, Textaufgaben und Programmieraufgaben, zu finden und umzusetzen. Grundlage bildete die Aufgabensammlung zur Vorlesung „Grundlagen der Informatik 1“.

Zunächst wurden die Aufgaben in Kategorien unterteilt, um für die unterschiedlichen Arten von Aufgabenstellungen und Lösungswege, Methoden zur Vorkorrektur finden zu können. Dabei wurden zwei Hauptkategorien „algorithmische Aufgaben“ und „Textaufgaben“ erstellt innerhalb derer dann nochmals feiner klassifiziert wurde.

Verschiedene bereits existierende Systeme und wissenschaftliche Beiträge wurden daraufhin im nächsten Schritt auf ihre Möglichkeiten und Ergebnisse bezüglich einer automatischen Korrektur untersucht. Außerdem wurden Methoden der Wissenschaft, insbesondere der Computerlinguistik, ermittelt, die später beim Erstellen des Systems umgesetzt wurden.

Im darauffolgenden Schritt wurde der eigene Ansatz formuliert und das Konzept des Systems, unter Berücksichtigung der Aufgabenstellung und der Recherche der bereits existierenden Systeme und Beiträge, sowie wissenschaftlicher Methoden, erstellt. Dabei wurde sowohl ein Konzept für die Kategorie „algorithmische Aufgaben“ sowie die Kategorie „Textaufgaben“ kreiert. Zudem wurde berücksichtigt, dass künftige Arbeiten auf dieses Konzept aufsetzen können und Daten zur automatischen Vorkorrektur zur Auswertung für maschinelle Lernverfahren vorliegen.

Das Konzept wurde durch die Entwicklung eines C#-Programms realisiert. Das Programm ist in der Lage verschiedenartige Aufgabentypen vorzukorrigieren und dem Tutor zu präsentieren. Es wurde eine Struktur in Form von Formularen innerhalb eines Word-Dokuments festgelegt, innerhalb derer die Aufgaben zu bearbeiten sind. Die Punktzahl, die der Tutor unter Berücksichtigung der Vorkorrektur festlegt, wird vom System wiederum in das eingesendete Dokument ausgeleitet.

Die Analyse des entwickelten Systems und die Auswertung der Ergebnisse fanden durch Beispielaufgaben und Beispiellösungen zu den beiden Kategorien „algorithmische Aufgaben“ und „Textaufgaben“ statt. Es wurde ermittelt inwieweit das System in der Lage ist die Aufgabetypen vorzukorrigieren und wieviel Aufwand für die Verwendung des Systems, die Auswertung der Vorkorrektur und die Nachkorrektur vom Anwender betrieben werden muss.

Die Arbeit zeigt, dass das automatische Korrigieren von Aufgaben ein äußerst umfangreiches und komplexes Thema ist. Es gibt eine Vielzahl verschiedener Aufgabetypen für die größtenteils bereits Lösungen existieren, diese sind wiederum jedoch sehr spezifisch. Eine allgemeingültige Lösung zu finden, die viele verschiedene Aufgabetypen auf einmal – ohne die Notwendigkeit von Erweiterung – abdeckt, erschien innerhalb dieser Arbeit als nicht möglich. Im letzten Abschnitt soll erläutert werden wie man zukünftig eine Verbesserung oder Erweiterung bei der Entwicklung eines automatisch korrigierenden Systems erreichen könnte.

7.1 Zukünftige Arbeit

Die hohe Komplexität, geschuldet durch die Vielzahl an unterschiedlichen Aufgabearten, macht es schwierig eine Lösung auszuarbeiten, die für alle möglichen Aufgabetypen angewandt werden kann. Um ein höheres Maß an algorithmischen Aufgaben abdecken zu können, könnte man Systeme verwenden oder erschaffen, die dem Programm Korrekturvorschriften beibringen, statt sie selbst zu programmieren. Weiterhin existieren würde dann jedoch das in Abschnitt 6.1.2 erläuterte Problem der fest vorgegebenen Struktur, die einen Teil der Lösungsfindung vorwegnimmt. Ein anderer Ansatz wäre, auch auf algorithmische Aufgaben Methoden der Textanalyse anzuwenden und somit dem Studenten die Freiheit zu lassen, inwiefern und in welcher Darstellungsform er seine Aufgabe löst.

Bei Freitext-Aufgaben könnte eine Anpassung der momentanen Berechnung der Korrekturwerte zu besseren Ergebnissen führen. So könnten existierende Faktoren anders und eventuell besser gewichtet werden. Zusätzliche Aspekte (beispielsweise die Häufigkeit des Vorkommens eines einzelnen Token) stellen ebenfalls Indizes für Übereinstimmung dar und könnten mit aufgenommen werden.

Am interessantesten wäre sicherlich die Integration eines maschinellen Lernverfahrens, mit dem man das System trainieren könnte. Auf Basis der gegebenen Antworten und vorliegenden Musterlösungen, sowie Entscheidungen des Tutors bezüglich der Bewertung von Abgaben könnte das System mit der Zeit erkennen, ob eine Abgabe als richtig oder falsch zu bewerten ist.

Mittel- und langfristig wäre höchstwahrscheinlich eine Zusammenarbeit mehrerer Hochschulen und eventuell auch anderer Bildungseinrichtungen äußerst hilfreich um die Arbeit an einem ausgereiften System zur automatischen Vorkorrektur zu forcieren und möglichst viele unterschiedliche Aufgabenarten abdecken zu können.

Die jetzige Arbeit fand innerhalb der Rahmenbedingung „eingesendete Aufgaben im Word-Format“ statt und wurde durch eine lokal ausführbare Windows-Applikation realisiert. Für die reine Arbeit an einem System zur automatischen Vorkorrektur und darauffolgender Analyse genügt das, für einen späteren Einsatz in einer realen Umgebung wäre jedoch beispielsweise die Integration in eine bestehende Lernmanagement-Plattform sinnvoll und zielführend. Somit könnten andere Internet-Dienste (Anbindung großer Datenbanken, Online-Sprachverarbeitung etc.) besser integriert und durch die Plattformunabhängigkeit und den hohen Gebrauch von Lernmanagement-Plattformen an Universitäten eine höhere Zielgruppe erreicht werden.

Danksagung

Mein besonderer Dank gilt Herrn Professor Roller für die Möglichkeit diese Arbeit an seinem Institut verfassen zu dürfen.

Bei meinen beiden Betreuern Felix Baumann und Julian Eichhoff bedanke ich mich recht herzlich für die Unterstützung über die gesamte Dauer meiner Arbeit und insbesondere die wichtigen Ratschläge in der Schlussphase.

Meiner Familie und meinen Freunden danke ich für die stetige Unterstützung während meines gesamten Studiums.

Literaturverzeichnis

[1] „Anwesenheitspflicht an der Uni - Uniturm Magazin,“ [Online]. Available: http://www.uniturm.de/magazin/recht/anwesenheitspflicht-an-der-uni-973. [Zugriff am 01 02 2015].

[2] H. Herold und B. Lurz, Grundlagen der Informatik (Pearson Studium - IT), München: Pearson Deutschland GmbH, 2012.

[3] „Computerlinguistik - LMU München,“ [Online]. Available: https://www.uni-muenchen.de/informationen_fuer/presse/expertenservice/c/computerlinguistik/index.html. [Zugriff am 07 02 2015].

[4] R. P. D. Manthey, „Uni Bonn - Vorlesung Informatik II (SS 02),“ [Online]. Available: http://www.iai.uni-bonn.de/III/lehre/vorlesungen/Informatik_II/SS02/folien/B9sw4.pdf. [Zugriff am 16 02 2015].

[5] „Taxonomie der kognitiven Lernziele,“ [Online]. Available: http://www.stangl.eu/psychologie/praesentation/lernziele.shtml. [Zugriff am 04 12 2014].

[6] „Sächsisches E-Competence-Zertifikat,“ [Online]. Available: http://www.seco-sachsen.de/fileadmin/upload/download_content/Handreichung_Formulierung%20von%20Lernzielen_secolayout_100302.pdf. [Zugriff am 08 02 2015].

[7] G. D. Rey, E-Learning, Theorien und Gestaltungsempfehlungen, Bern: Verlag Hans Huber, 2009.

[8] „CampusSource,“ [Online]. Available: https://www.campussource.de/wir/. [Zugriff am 17 02 2015].

[9] „CampusSource - Software - Ilias,“ [Online]. Available: https://www.campussource.de/software/ilias/. [Zugriff am 03 12 2014].

[10] „moodle.org: Moodle Statistics,“ [Online]. Available: https://moodle.net/stats/. [Zugriff am 04 02 2015].

[11] „ILIAS E-Learning - ILIAS Features,“ [Online]. Available: http://www.ilias.de/docu/ilias.php?ref_id=392&from_page=16839&obj_id=11674&obj_type=StructureObject&cmd=layout&cmdClass=illmpresentationgui&cmdNode=b0&baseClass=ilLMPresentationGUI. [Zugriff am 03 12 2014].

[12] Universität Wien - Zentraler Informatikdienst, „MoodleTreff,“ März 2010. [Online]. Available: https://www.moodletreff.de/file.php/211/Dokumentationen/Moodle_Aufgaben.pdf. [Zugriff am 04 02 2015].

[13] „Funktionsumfang ausgewählter E-Prüfungssysteme - E-Assessment-Wiki,“ [Online]. Available: http://ep.elan-ev.de/index.php?title=Funktionsumfang_ausgew%C3%A4hlter_E-Pr%C3%BCfungssysteme. [Zugriff am 03 12 2014].

[14] S. J. Gruttmann, *Formatives E-Assessment in der Hochschullehre - Computerunterstützte Lernfortschrittskontrollen im Informatikstudium,* Paderborn: Westfälische Wilhelms-Universität Münster, 2009.

[15] „PRAKTOMAT,“ [Online]. Available: https://pp.ipd.kit.edu/projects/praktomat/praktomat.php. [Zugriff am 07 02 2015].

[16] „Automatische Korrektur von Programmieraufgaben - Ein Erfahrungsbericht --- Forschung - Fakiltät Informatik, Mathematik und Naturwissenschaften,“ [Online]. Available: https://portal.imn.htwk-leipzig.de/fakultaet/weicker/forschung/download/eclaus3.pdf. [Zugriff am 07 11 2014].

[17] „Institut für Wirtschaftsinformatik, Praktische Informatik, Universität Münster,“ [Online]. Available: http://www.wi1.uni-muenster.de/pi/lehre/ss08/SeminarE-Learning/ausarbeitungen/ausarbeitung_wingender.pdf. [Zugriff am 07 11 2014].

[18] K.-U. Carstensen und C. Ebert, Computerlinguistik und Sprachtechnologie: Eine Einführung, Heidelberg: Spektrum Akademischer Verlag, 2010.

[19] „Hochschule Osnabrück: Themenbereich IV: Natural Language Processing, Sentiment-Analyse,“ [Online]. Available: http://www.ecs.hs-osnabrueck.de/44188.html. [Zugriff am 06 02 2015].

[20] „ILIAS E-Learning - Installation and Maintenance,“ [Online]. Available: http://www.ilias.de/docu/goto_docu_pg_6531_367.html. [Zugriff am 21 02 2015].

[21] D. Frischalowski, Visual C# 2010 - Einstieg für Anspruchsvolle, München: Pearson Studium, 2010.

[22] R. Wenger, Handbuch der .NET 4.0-Programmierung. Band 1: C# 2010 und .NET-Grundlagen, Köln: O'Reilly Verlag GmbH & Co. KG, 2010.

[23] „Verfügbarkeit von Funktionen in Visual Studio-Versionen," [Online]. Available: https://msdn.microsoft.com/de-de/library/ee519072.aspx. [Zugriff am 08 02 2015].

[24] „Übersicht über Windows Forms," [Online]. Available: https://msdn.microsoft.com/de-de/library/8bxxy49h(v=vs.110).aspx. [Zugriff am 08 02 2015].

[25] W. G. Stock, Information Retrieval: Informationen suchen und finden, München: Oldenbourg Wissenschaftsverlag GmbH, 2007.

[26] J. Lang und R. Kowatschew, 15 04 2015. [Online]. Available: http://kontext.fraunhofer.de/haenelt/kurs/Referate/Kowatschew_Lang/stemming.pdf.

[27] „HRG - Einzelnorm," [Online]. Available: http://www.gesetze-im-internet.de/hrg/__15.html. [Zugriff am 20 04 2015].

[28] „WGD HTML Validator Changelog," [Online]. Available: http://www.htmlhelp.com/tools/validator/changelog.html.en. [Zugriff am 02 02 2015].

[29] „Object Linking and Embedding - Wikipedia," [Online]. Available: http://de.wikipedia.org/wiki/Object_Linking_and_Embedding. [Zugriff am 04 02 2015].

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

(Ort, Datum, Unterschrift)