INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Projektarbeit

Erstellung und strukturelle Aufbereitung des Informatik II Repositorys

Entwurf eines Verwaltungssystems & Übernahme der bestehenden Aufgabe in das neue System

Jörg Schmidl

Aufgabensteller: Prof. Dr. Hans Jürgen Ohlbach
Betreuer: Dr. Norbert Eisinger, Bernhard Lorenz
Abgabetermin: 23. August 2005

Inhaltsverzeichnis

1 Aufgabenstellung und Zielsetzung

1.1 Über die Vorlesung Informatik 2

Die Vorlesung Informatik 2 ist Teil des Pflicht-Programms im Informatik-Grundstudium an der LMU München. Sie findet zur Zeit in jedem Sommersemester als vierstündige Vorlesung statt, ergänzt um zweistündige Präsenzübungen (Tutorien) in kleinen Gruppen. Die Vorlesung richtet sich an Studenten der Informatik im zweiten Fachsemester. Ihr Schwerpunkt liegt auf der Darstellung der Konzepte objektorientierter Programmiersprachen. Dazu wird zur Zeit die Programmiersprache Java für die praktischen Übungen eingesetzt.

1.2 Übungsbetrieb

1.2.1 Allgemeiner Ablauf des Übungsbetriebs

Im Laufe des Semesters werden zwischen zehn und zwölf Übungsblätter ausgegeben (pro Woche eines), mit denen der Vorlesungsstoff anhand praktischer Aufgaben vertieft wird. Jedes Übungsblatt wird in einem zweistündigen Tutorium besprochen. Die aktive Teilnahme am Übungsbetrieb ist Voraussetzung für die Zulassung zur Abschlussklausur am Semesterende. Auf jedem Übungsblatt sind einige Aufgaben als Hausaufgaben gekennzeichnet. Diese müssen von den Studenten selbständig gelöst und wöchentlich abgegeben werden.

1.2.2 Erstellung der Übungsblätter

Da die Vorlesung abwechselnd von unterschiedlichen Dozenten gehalten wird und sich damit meist auch der Fokus auf die verschiedenen Themen ändert, sind auch Gliederung und Inhalt manchmal sehr verschieden. Aktuelle Erweiterungen in der verwendeten Programmiersprache JAVA fließen soweit wie möglich in die Übungen ein. Daher werden alle Übungsblätter in jedem Semester neu erstellt, wobei selbstverständlich zu einem gewissen Teil Aufgaben früherer Semester (in leicht veränderter Form) wiederverwendet werden. Jedes Übungsblatt ist prinzipiell eine **Zusammenstellung** dieser Art:

- Vorlesungsparameter (Semester, Dozent, Abgabetermin usw.)
- Hinweise und aktuelle Informationen
- die eigentlichen Aufgaben

Ein Übungsblatt in seiner unkompilierten Version ist eine Komposition von Latex-Quelltexten der einzelnen Aufgaben. Desweiteren werden teils die Programmtexte der praktischen Programmieraufgaben oder Diagramme inkludiert.

1.3 Diese Projektarbeit

Aufgabe und Zielsetzung dieser Arbeit ist primär die Erstellung eines zentralen Verwaltungssystems um eine vollständige **strukturelle, thematische und inhaltliche** Zusammenstellung der bisher ungeordneten Aufgaben für den Übungsbetrieb zur Vorlesung Informatik 2 zu ermöglichen. Damit einher geht eine Vereinheitlichung der Konzepte zur **Versionierung, Archivierung und Benennung** im neuen System.

1.3.1 Zur Struktur dieser Projektarbeit

Zunächst wird der Ist-Zustand der bisherigen Übungsverwaltung analysiert und bewertet. Aufbauend darauf werden die Rahmenbedingungen für die Gestaltung und Implementierung des Verwaltungssystems formuliert und in ein Konzept umgesetzt. Dann folgt eine Bedienungsanleitung für das erstellte System. Der nächste Abschnitt geht auf die Konventionen ein, die für die übernommenen Aufgaben getroffen wurden. Für Weiterentwickler ist der nächste Abschnitt besonders interessant, da in ihm eine eingehende Beschreibung der technischen Implementierung enthalten ist. Abschließend werden einige Verbesserungsvorschläge und Ausblicke gegeben.

1.3.2 Ergebnis

Bei Abgabe der Projektarbeit, besteht ein voll funktionsfähiges System, welches in JAVA implementiert wurde und eine (Standard-SQL-) Datenbank als persistenten Datenspeicher nutzt. Alle gewünschten Anforderungen wurden erfüllt. Das System durchlief mehrere Entwicklungsiterationen und wurde ausgiebig getestet. Des weiteren wurden im Rahmen der Projektarbeit 116 Übungsaufgaben aus drei Semestern in das erstellte Übungssystem integriert und somit eine solide Datenbasis geschaffen.

2 Bisheriger Zustand des Aufgaben-Fundus

Um das gewünschte System entwerfen zu können war natürlich die eigene Erfahrung als Student hilfreich. Allerdings musste für eine umfassendere Beurteilung der Anforderungen an das System eine Betrachtung und Bewertung der bisher bestehenden Aufgaben geschehen. Beim Betrachten der bisherigen Aufgaben wurden die im folgenden genannten Missstände aufgedeckt.

2.1 Wiederverwenbarkeit

Bisher gibt es keine zentrale Sammlung für Aufgaben. Für jede Veranstaltung (also im Regelfall alle zwei Semester für die gleiche Vorlesung) wird mit einem “leeren Repository“ begonnen. Natürlich kann man, sofern der Lehrstuhl Zugriff darauf hat, die Übungsaufgaben der letzten Semester als Ausgangspunkt nutzen. Dabei gibt es aber Schwierigkeiten. Die Vorlesung wird im Allgemeinen nicht im zwei-Semester Turnus vom selben Lehrstuhl gehalten, sondern wird im Gegenteil pro Jahr von jeweils einem anderen Lehrstuhl durchgeführt. Da man nicht ohne weiteres Zugriff auf die Übungsblätter anderer Lehrstühle hat, ist die Menge der Vorlagen dadurch beschränkt. Auch wenn man Zugriff auf die jeweiligen Aufgabensammlungen bekommt besteht noch immer das Problem, dass die Formate im Allgemeinen nicht einheitlich sind. Sie können sich auch im Laufe der Zeit innerhalb eines Lehrstuhl geändert haben. Dadurch ist eine einfache Übernahme und ein einheitlicher Zugriff auf alte Aufgaben nicht gewährleistet.

2.2 Versionierung

Es gibt bisher keine Möglichkeit eine Versionierung durchzuführen und auch ersichtlich zu machen. Es ist daher nicht oder nur umständlich möglich Aufgaben auf dem neuesten Stand zu halten.

2.3 Aufgabe und Lösung

Aufgaben wurden bisher meist mit Lösung abgespeichert, wodurch sich eine Vermischung zweier unterschiedlicher logischer Einheiten ergiebt. Es gibt auch Fälle in denen die Lösung separat ist, aber dann kein Bezug zwischen Angabe und Lösung besteht, abgesehen von entweder Namenskonventionen oder Ordnerstrukur. Beide Varianten sind unzureichend für ein möglichst flexibles System, da es keine allgemeine strukturelle Verknüpfung zwischen Angabe und Lösung einer Übungsaufgabe gibt und somit die Wartbarkeit erschwert wird.

2.4 Logische Beziehungen

Es gibt keine Unterstützung logische Beziehungen zwischen verschiedenen Aufgaben zu modellieren.

2.5 Suche

Es gibt keine komfortable Suchmöglichkeit, weil die Aufgabensammlung eines Semesters bisher ohne indexierbare Struktur abgelegt ist. Will man eine Aufgabe als Vorlage nutzen so muss man also viele Übungsblätter durchsuchen um vielleicht eine passende zu finden. Aus diesem Umstand heraus ist es leider auch nicht möglich aus den Erfahrungen früherer Semester vollen Wert zu schöpfen.

2.6 Bewertung

Zusammenfassend ist der aktuelle Aufgabenbestand ein nur schwer nutzbarer Pool von Aufgaben, die aus verschiedenen Semestern und auch von zwei verschiedenen Lehrstühlen stammt. Die Aufgaben sind in keiner übergeordneten Struktur und ein Zusammenhang zwischen Aufgaben ist nicht intuitiv ersichtlich. Die Möglichkeit passende Aufgaben zu finden hängt also stark von der Einarbeitung des Aufgabenerstellers (oder Erfahrung falls die Aufgabe in früheren Semestern bereits übernommen wurde) ab.

3 Entwurf des Aufgabenverwaltungssystems

Aus den weiter oben beschriebenen Missständen kann man einige Anforderungen an das zu entwicklende System sofort ableiten. Andere ergeben sich durch spezielle Anforderungen der Vorlesung Informatik 2.

3.1 Anforderungen

3.1.1 Aus Missständen abgeleitete Anforderungen

- Das System soll die bisher genutzten Aufgaben enthalten.
- Das System soll eine strukturierte, geordnete Zusammenstellungen von potentiellen Übungsaufgaben bzw. Vorlagen für neue Übungsaufgaben darstellen.
- Die Daten des Systems sollen zentral gehalten werden, um einen einheitlichen Zugriff zu erleichtern.
- Eine Unterstützung für Versionierung sowie für die Bildung von Varianten soll gegeben sein.

- Die Angabe und die Lösung sollen gemeinsam referenzierbar aber getrennte Einheiten sein.
- Beziehungen zwischen Aufgaben sollen modelliert werden können.
- Eine komfortable, möglichst umfassende Suchfunktion soll bereitgestellt werden.

3.1.2 Von Vorlesungsspezifika abgeleitete Anforderungen

- Da die Angaben und Lösungen durchaus aus mehreren (Programmcode) Dateien bestehen können, müssen diese in einem Ordner erstellt werden können.
- In JAVA wird die Package Struktur im Dateisystem durch eine hierarchische Ordnerstruktur abgebildet. Es muss deshalb eine solche sowohl für Angabe als auch für Lösung bereitgestellt werden können.
- Einzelne Programmcode Dateien können zu verschiedenen Aufgaben gehören (insbesondere wenn sie relativ allgemeinen Inhalt haben). Da eine einfache Duplizierung des Programmcodes die Versionierung, Variantenbildung und Verwaltung im Allgemeinen unnötig erschwert, soll ein Verweismechanismus genutzt werden.
- Aufgaben können einen Wissensstand als Voraussetzung haben, ohne den die Bearbeitung der Aufgabe wenig sinnvoll ist. Es sollte darum möglich sein die Aufgaben mit dieser Information auszustatten.
- Feedback von Betroffenen (Studenten wie auch Tutoren) kann hilfreich sein für die Weiterentwicklung der Aufgaben. Es wäre deshalb gut dieses Feedback den Aufgaben anhängen zu können.
- Es gibt verschiedene Themengebiete, denen die Aufgaben zugeordnet werden können. Diese Information ist wichtig, wenn man nach Aufgaben suchen muss.
- Aufgaben können aufeinander aufbauend sein. Daher muss eine Reihenfolgebeziehung modellierbar sein.
- Es gibt in den Übungsaufgaben häufig den Fall, dass ein Programmrahmen zu vervollständigen ist. Um die Angabe und die Lösung nicht getrennt warten zu müssen, bietet es sich an die Lösung mit Exkludierung der jeweiligen Lösungsbestandteile als Angabe zu nutzen.

3.2 Entwurf

3.2.1 Technisch

Um die oben gestellten Anforderungen erfüllen zu können gibt es verschiedene Ansätze. Wie in anderen Vorlesungen üblich könnte man die Verwaltung über das Dateisystem bewerkstelligen. Dies würde allerdings erstens eine Platformabhängigkeit einführen (was in der Praxis wohl weniger schlimm wäre), und zweitens ist die Erweiterbarkeit unter Umständen weniger einfach. Der passendere Ansatz ist in diesem Fall die Nutzung einer Datenbank. Diese lässt sich platformunabhängig einsetzen und bietet eine gute Erweiterbarkeit. Daher wurde für diese Projektarbeit die OpenSource Datenbank MySQL genutzt. Um auf die Daten komfortabel zugreifen zu können ist eine grafische Nutzerschnittstelle angebracht. Diese kann prinzipiell in jeglicher Programmiersprache erstellt werden. Die persönlichen Präferenzen des Autors haben zur Wahl von JAVA geführt. Die erstellte Oberfläche ist eine JAVA Swing

Anwendung. Sie bietet eine komfortable Möglichkeit um Aufgaben verwalten zu können. Alle oben gennanten Anforderungen werden erfüllt. Die Nutzung des Systems wird Thema des nächsten Abschnitts sein. Eine detailierte, technische Beschreibung wird im vorletzten Kapitel gegeben.

3.2.2 Inhaltlich

Die im folgenden gezeigte, generelle Sruktur wurde identifiziert und implementiert:

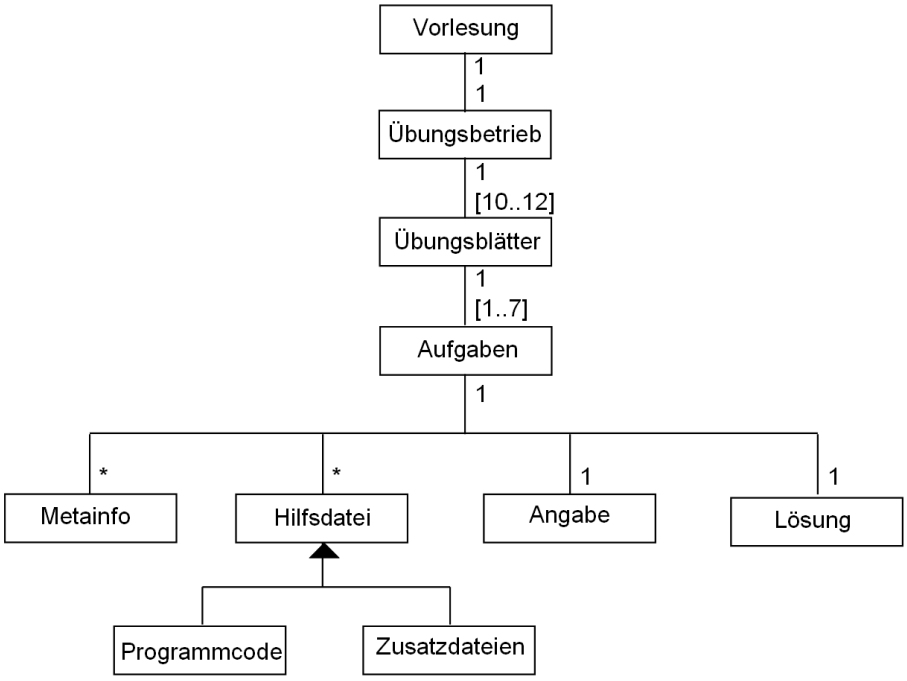


Abbildung 1: Zusammenhang der Systembestandteile

Eine **Vorlesung** hat also einen Übungsbetrieb und einige weitere Elemente die aber für die Übungsverwaltung unwichtig sind. Für die Applikation werden bloß Parameter wie z.B. der Name der Vorlesung, interessant sein.

Der **Übungsbetrieb** besteht aus (meist ca.) zehn bis zwölf Übungsblättern und weiteren Informationen, wie einem Standardausgabeverzeichnis, dem genutzten Latex-Package zur Übungsblatterstellung und anderen, für die Applikation weniger wichtigen, Elementen.

Ein **Übungsblatt** hat eine laufende Number, eventuell eine Mitteilung an die Studenten und natürlich eine Komposition von Aufgaben.

Aufgaben sind das zentrale Element des Verwaltungssystems. Sie bestehen nach aussen hin aus einem Angabentext, einer meist intern genutzten Musterlösung sowie, je nach Aufgabe, einer Reihe von Dateien. Letztere sind meist Programmrahmen, können aber auch Diagramme oder HTML-Dateien sein, je nach verfolgtem Lernziel. Selbstverständlich gibt es in solchen Fällen dann auch entsprechende Dateien für die Musterlösung. Weniger of-

fensichtlich haben Aufgaben auch Metainformationen, also Informationen die eine Kategorisierung der Aufgabe erlauben. Die in der Applikation unterschiedenen und verwendeten Metainformationen werden weiter unten in diesem Abschnitt aufgeführt und erklärt.

Hilfsdateien stellen in erster Linie Dateien beliebiger Art dar, die in einer Angabe oder Lösung genutzt werden, sei es durch Bereitstellen der Dateien in der Angabe der Übungsaufgabe, als interne Musterlösung oder um sie in den Latex-Quellcode zu inkludieren. Es werden im System jedoch zwei Arten von Dateien unterschieden. Zum einen die **Programm-Code** Dateien und zum anderen die **Zusatzdateien**. Programm-Code Dateien enthalten Programm-Code der genutzten Programmiersprache (also JAVA) der entweder bereits kompilierbar ist oder aber von den Studenten vervollständigt werden soll. Zusatzdateien sind alle anderen Dateien. Vordergründig sind es Textdateien die Programmausgaben oder Testdaten enthalten, oder aber (UML-) Diagramme. Grund für die getroffene Unterscheidung war die Anforderung möglichst die Exkludierung bestimmter Programm-Code-Teile zu unterstützen. Um dies bewerkstelligen zu können, muß ein Programm-Code deshalb geparst werden, andere Zusatzdateien müssen dies jedoch nicht. Ebenso kann ein Programm-Code als Kompilat oder Klartext vorkommen, während eine Zusatzdatei immer nur eine Form hat.

Sowohl **Angabe** als auch **Lösung** enthalten Latex-Quellcode. Bei beiden können Inkludierungen von Dateien auftreten.

Folgend sind die im System unterschiedenen Metainformationen aufgeführt.

Thema
Das Thema einer Aufgabe gibt dessen grobe inhaltliche Kategorie wieder. So ist beispielsweise “Objektorientierung“ ein Themengebiet. Wenn es angebracht ist ein Thema genauer zu untergliedern, kann man auch Unterthemen anlegen. “Objektorientierung“ allein ist zum Beispiel relativ grob, da ein größerer Teil der Vorlesung sich mit diesem Thema beschäftigt. Es bieten sich hier also Unterthemen wie etwa “Dynamisches Binden“ an um als Thema der Aufgabe zu fungieren.

Fokus
Der Fokus einer Aufgabe beschreibt, auf welcher Ebene ein Thema eingeführt werden soll. So kann es sein, dass eine Aufgabe ihren Fokus auf der Syntax eines JAVA Konstrukts hat, während eine andere Aufgabe die Syntax als gegeben oder weniger wichtig erachtet und lieber die spezifischen Ideen der JAVA API herausstellen will. Beispielsweise könnte eine Aufgabe zu Arrays darin bestehen syntaktische Fehler zu finden um so die Nutzung von Arrays in Java einzuüben, während eine andere Aufgabe zu Arrays die Betonung auf die Möglichkeiten der API-Klasse Arrays legt um die Nützlichkeit von Arrays in JAVA zu untermauern.

Form
Die Form gibt an wie eine Aufgabe zu lösen ist. Die meisten Aufgaben dieser Vorlesung werden sich mit Programmierung, also mit eigenständigem Programmieren oder dem Vervollständigen von Programmrahmen beschäftigen. Es gibt jedoch noch eine Hand voll anderer Formen, wie Freitextantworten oder Fehlersuche.

Komplexität
Die Maßzahl Komplexität versucht, mit ihrer momentan numerischen, fünf-stufigen Skala, so gut wie möglich die schwer zu beschreibende Eigenschaft, wie “schwierig“ eine Aufgabe

ist wieder zu geben.Die Zahl 1 soll hierbei einfach bedeuten, 3 moderat schwer und 5 relativ schwer. Ein numerischer Wert ist leider extrem abstrakt und die genaue Zuordnung nicht immer klar. Andere Einordnunugen wurden angedacht, schienen aber auch nicht geeigneter.

Weitere Merkmale

- Auf jeder Hierarchiestufe (also Übungsblatt, Aufgabe und Zusatzdatei) ist es möglich einen internen Kommentar hinzu zufügen. Dies ermöglicht eine feingranulare, retrospektive Information über die Qualität des jeweiligen Elements.
- Aufgaben wie auch Zusatzdateien haben eine Versionsnummer und eine Variantennummer.
- Aufgaben können thematische Voraussetzungen zugewiesen werden.
- Eine Einorndung einer Aufgabe in eine logische Aufgabereihenfolge wird unterstützt.
- Es wird überprüft und angezeigt wann und wie oft eine bestehende Aufgabe bereits benutzt wurde.
- Es werden numerische Werte hinsichtlich des Umfangs einer bestehenden Aufgabe angezeigt.
- Die Suche nach Aufgaben wird nach allen vorhandenen Kriterien unterstützt.
- Es wurde darauf geachtet die verschiedenen Eingabeformulare homogen zu gestalten um die Nutzung zu erleichtern.

Variante vs. Version
Diese Differenzierung macht Sinn, um folgendes zu gewährleisten: Bei verschiedenen Versionen einer Aufgabe soll davon ausgegangen werden, dass immer die letzte Version auch die aktuellste ist und damit allen Vorgängerversionen vorzuziehen ist. Verschiedene Varianten hingegen können sich in ihrer Aufgabenstellung (in gewollter Weise) leicht unterscheiden um so eine bessere Wiederverwendbarkeit der Aufgaben zu gewähleisten.

4 Nutzung des Verwaltungssystems

4.1 Installation

Da die Anwendung JAVA-basiert ist wird eine installierte **JAVA Runtime Environment** in der Version 5.0 vorausgesetzt. Desweiteren wird eine **Datenbank** benötigt. In der derzeitigen Form nutzt die Applikation standardmäßig eine MySQL-Datenbank. Es wurden allerdings nur Standard-SQL Konstrukte verwenden. Somit sind also alle gängigen Datenbanken ebenso nutzbar. Die im Quelltext referenzierte Datenbank heißt **aufsam**. Die Datenbank im Datenbankverwaltungssystem sollte also ebenso heißen. Ist dies nicht gewünscht kann man in einer Konfigurationsdatei die jeweils gewünschte Datenbankumgebung eintragen. Näheres dazu in der zugehörigen README. Im äußersten Fall kann der Datenbankzugriff auch leicht im Quelltext geändert werden, da alle Konstanten an einer Stelle im Quelltext gehalten werden. Sollte die verwendete Datenbank geändert werden, darf man nur nicht vergessen noch den passenden Datenbanktreiber für JAVA in das lib Verzeichnis zu legen (standardmäßig liegt dort nur der Datenbanktreiber für MySQL). Bei Änderungen am Quelltext ist ein Neukompilieren von Nöten. Nutzt man MySQL mit “aufsam“ als Datenbank so ergeben sich

lediglich folgende Arbeitsschritte:
MySQL Administration öffnen.
“create databse ausfsam“ eingeben.
“use aufsam“ eingeben.
“source (Pfad zur mitgeliefert aufsamDb.sql Datei)“ eingeben.
Fertig.

4.2 Verwendung

Ist die Datenbank wie in der mitgelieferten Konfigurtionsdatei eingerichtet, reicht ein ausführen des mitgelieferten JAR-Archivs. Sollte die Datenbank anders eingerichtet sein, muss zudem als Parameter der absolute Pfad der Konfigurationsdatei angegeben werden. Letzlich kann das Archiv natürlich zur Nutzung entpackt werden. In diesem Fall wird die Applikation durch den Aufruf **java de.lmu.ifi.pms.aufsam.Starter** gestartet. Man bekommt folgende Oberfläche zu sehen:

Abbildung 2: Startansicht der Applikation

Die rechte Seite der Oberfläche stellt die Eingabeformulare dar bzw. präsentiert die gewünschten Informationen. Anfänglich sieht man hier nur ein Willkommensfenster. Auf der linken Seite ist die Navigation angebracht. Hier kann man die verschiedenen Möglichkeiten des Systems nutzen:

- **Aufgabe suchen**
  Durch drücken dieses Knopfes wird eine Suchmaske geöffnet die es gestattet eine Aufgabe unter Verwendung verschiedener Filterkriterien zu suchen.
- **Aufgabe erstellen**
  Dieser Knopf öffnet eine Eingabemaske die strukturell mit der Suchmaske nahezu identisch ist, aber zur Neueingabe einer Aufgabe dient.
- **Übungsblatt erstellen**
  Ein neues Übungsblatt kann durch drücken dieses Knopfes zusammengestellt werden.

- **Übungsblatt betrachten**
  Bereits im System befindliche Übungsblätter können durch drücken dieses Knopfes angezeigt werden.
- **Hilfsdatei in DB speichern**
  Das Anlegen einer neuen Hilfsdatei (Programm-Code Datei oder Zusatzdatei) geschieht über das Betätigen dieses Knopfes.
- **Hilfsdatei ändern**
  Durch drücken dieses Knopfes erhält man eine Auflistung aller in der Datenbank gespeicheten Hilfsdateien. Diese können dann editiert werden um so eine neue Version oder Variante anzulegen.
- **Vorlesungsprofil erstellen**
  Ein Vorlesungsprofil fasst Merkmale einer Vorlesung zusammen und kann durch drücken dieses Knopfes erstellt werden.

4.2.1 Aufgaben suchen

Das Suchformular
Das Suchformular wird durch die Navigation im linken Teil der Oberfläche geöffnet. Es ist grafisch (durch Rahmen) in verschiedene Abschnitte strukturiert. Der oberste Abschnitt beherbergt eine Eingabemöglichkeit für den Titel einer Aufgabe. Dieser kann entweder exakt bestimmt sein oder mit einem SQL-Wildcard (% mit Bedeutung "beliebig viele Zeichen") versehen sein. Bleibt der Titel leer so wird er nicht als Suchkriterium genutzt. Im selben Abschnitt kann man auch nach einer bestimmten Version oder Variante suchen.

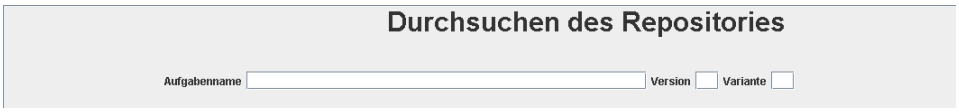


Abbildung 3: Titlel-Suchoption

Der darauf folgende Abschnitt enthält Auswahlmöglichkeiten für die Metainformationen einer Aufgabe, also dem Thema, dem Fokus, der Form und der Komplexität (zur jeweiligen Bedeutung siehe Kapitel **Entwurf des AufgabenVerwaltungssystems**) um diese als Suchkriterien zu nutzen. Die Auswahl ist hier jeweils durch ein Auswahlfeld realisiert. Werden die voreingestellten Werte genutzt (also leere Einträge) so wird das jeweilige Kriterium folgerichtig nicht in die Suche mit aufgenommen.

Abbildung 4: Metainformation-Suchoption

Als nächster Abschnitt kommt der Angaben- und der Antworttext. Bei beiden bietet es sich an mit dem SQL-Wildcard (%) zu arbeiten, da in den seltensten Fällen genau nach einem bestimmten Latex-Quellcode gesucht wird. Auch für die beiden Texteingaben gilt, dass sie nicht als Suchkriterium aufgenommen werden, wenn sie leer sind.

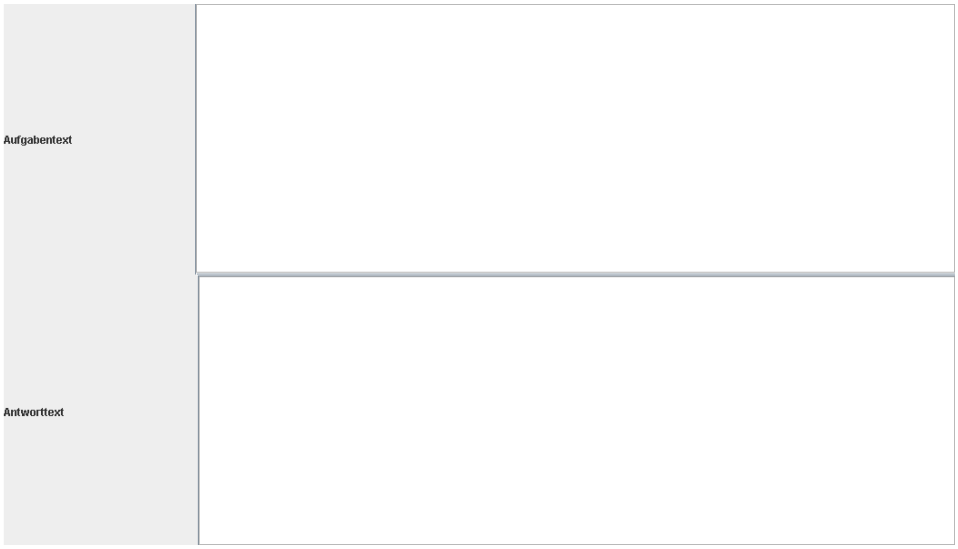


Abbildung 5: Angabe/Antwort-Suchoption

Die Angabe von genutzten Programm-Code Dateien bzw. Zusatzdateien kann im nächsten Abschnitt geschehen. Sind in diesen Listen Dateien spezifiziert, so werden nur die Aufgaben bei der Suchergebnisliste berücksichtigt, die auch diese Dateien nutzen. Leere Listen entsprechen wiederum einer Nichtnutzung dieser Suchopition. Die Abbildung zeigt das grafische Element für einen Programm-Code. Das korrespondierende Element für die Zusatzdatei sieht analog aus.

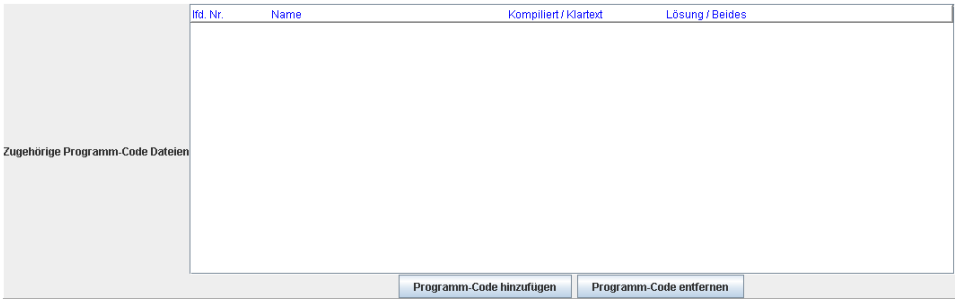


Abbildung 6: Hilfsdatei-Suchoption

Im Feld “Internes Feedback“ des nächsten Abschnitts kann nach Kommentarfragmenten (unter Nutzung des %-Wildcards) gesucht werden. Diese, auf den ersten Blick unnütze Funktion, kann gut genutzt werden um sich selbst eine Markierung zu setzen wenn man im Prozess der Aufgabenerstellung ist.

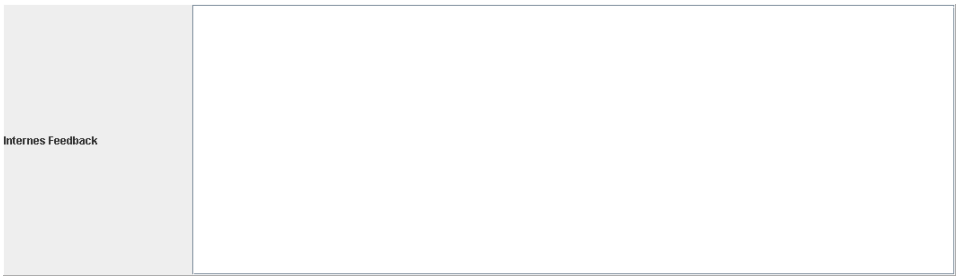


Abbildung 7: Internes-Feedback-Suchoption

Im vorletzten Abschnitt ist das Filtern nach Aufgaben, die bestimmte Themengebiete als Voraussetzung haben, möglich.

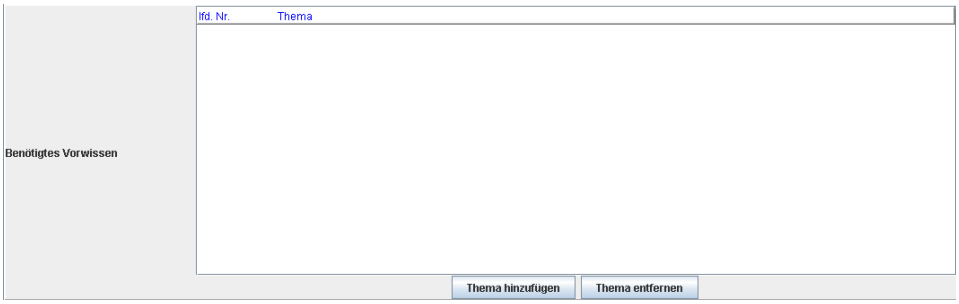


Abbildung 8: Themenvoraussetzung-Suchoption

Der letzte Abschnitt kann schließlich genutzt werden um die direkten Vorgänger bzw. Nachfolger einer Aufgabe als Suchkriterium zu nutzen. Dies kann sich als praktisch erweisen, wenn man sich entschlossen hat eine Aufgabe zu nutzen und man möchte einen roten Faden für das nächste Übungsblatt haben.

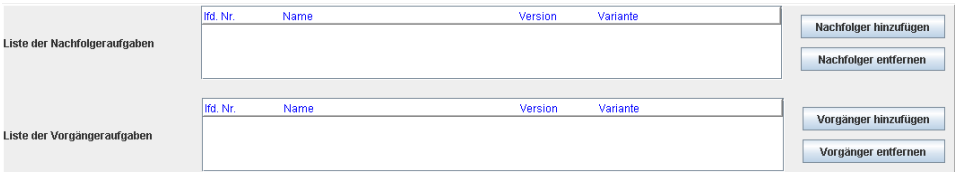


Abbildung 9: Vorgänger/Nachfolger-Suchoption

Hat man die gewünschten Suchkriterien wie eben beschrieben gewählt, so erscheint nach drücken des “**Suchen**“-Knopfes die Ergebnisliste. Sollte man die Suche doch nicht durchführen wollen, kann man durch drücken des Knopfes “**Abbrechen**“ die Suchmaske komplett schließen.

Die Ergebnisansicht
Hat man sich zum Durchführen der Suche entschlossen (also den Suchknopf gedrückt), öffnet sich ein neuer Tab, der die Ergebnisse der Suche in Form einer Liste anzeigt. In ihr sieht man die nötigsten Informationen einer Aufgabe, also die jeweilige laufende Nummer der Aufgabe, den Namen der Aufgabe, sowie die Version- bzw. Variantennummer der Aufgabe. Man kann nun diesen Tab wiederum durch drücken des Knopfes "**Abbrechen**" schließen, wenn die so erreichte Information schon aussreichend ist oder doch nicht dem erwarteten Ergebnis entspricht. Sollte man nähere Informationen zu einer der Aufgabe sehen wollen so kann man diese selektieren und mittels des Knopfes "**Auswählen**" unten auf der Ergebnisseite anzeigen lassen. Eine Selektion von mehr als einer Ergebnisaufgabe gleichzeitig ist möglich. Wie diese Mehrfachselektion bewerkstelligt wird ist platformabhängig, meist ist es aber durch auswählen des Datensatzes bei gleichzeitigem drücken der Steuerungstaste. Bei der Mehrfachselektion wird je ein Tab pro ausgewählter Aufgabe geöffnet. Eine exemplarische Ergebnisansicht mit Mehrfachselektion ist in folgender Abbildung zu sehen.

lfd. Nr.	Name	Version	Variante	Thema
01	AVLBaum	0	0	Collections
02	Abstrakte Klasse versus Interface	0	0	Vererbung
03	Abstrakte Klassen	0	1	Vererbung
04	Addition von Binärzahlen	0	0	Imperativ Programmieren
05	Applet-Lebenszyklus	0	0	Applets
06	Applets und Grafik	0	0	Applets
07	Arithmetik	0	0	Objektorientierung
08	Arrays	0	0	Imperativ Programmieren
09	Arrays	0	1	Imperativ Programmieren
10	Arrays/Vectors	0	0	Collections
11	Arten von Variablen	0	0	Java-Grundlagen
12	Arten von Variablen	0	1	Java-Grundlagen
13	Aufruf von Jar	0	0	Java Tools
14	Aufzählungen (enum)	0	0	Java-Grundlagen
15	Darlehen	0	0	Imperativ Programmieren
16	Darlehen als Applet	0	0	Applets
17	Darstellungen von Werten	0	0	Basistypen
18	Das kanonische erste Beispiel	0	0	Java-Grundlagen
19	Das kanonische erste Beispiel mit Fehlern	0	0	Java-Grundlagen
20	Datei durchsuchen	0	0	I-O-Ströme
21	Datei-Ein-/Ausgabe, Stringverarbeitung	0	0	I-O-Ströme
22	Datum.java	0	0	Stringverarbeitung
23	Dokumentation mit javadoc	0	0	Java Tools
24	Doppelt verkettete Listen	0	0	Collections

Abbildung 10: Suchergebnis

Die Detailansicht
Hat man sich entschlossen nähere Informationen zu einer oder mehreren Aufgaben anzuzeigen, so geschieht dies in einer Detailansicht. In dieser sind alle Felder die man in der Suchmaske füllen konnte, mit den jeweiligen Werten der Aufgabe gefüllt. Alle Eingabemöglichkeiten sind deaktiviert. Des weiteren gibt es einen zusätzlichen Abschnitt der numerische Informationen über die Aufgabe enthält. Zu diesen zählen die LOC (Lines of Code) der Angabe und der Lösung, wobei LOC nur von Program-Code Dateien gezählt werden. Ebenso enthalten ist die Anzahl der imports in den Program-Code-Dateien der Angabe sowie der

Lösung, um eine Abschätzung der Komplexität anzubieten (je mehr imports, desto komplexer). Gezählt werden hierbei aber nur imports die nicht aus der JAVA API direkt stammen (also nicht mir java. oder javax. beginnen). Die Wortanzahl der Angabe sowie der Lösung wird ebenso wiedergegeben. Hierbei ist es allerdings nur eine vage Approximation, weil nicht zwischen Nutzdaten (also eigentlichem Text) und Latex-Code unterschieden wird. Letztlich gibt es noch eine Liste der bisherigen Verwenug der Aufgabe, um eine Abschätzung treffen zu können, wie gut "wiederverwendbar" eine Aufgabe noch ist. Ein Beispiel für eine solche zusätzliche Information liefert die folgende Abbildung.

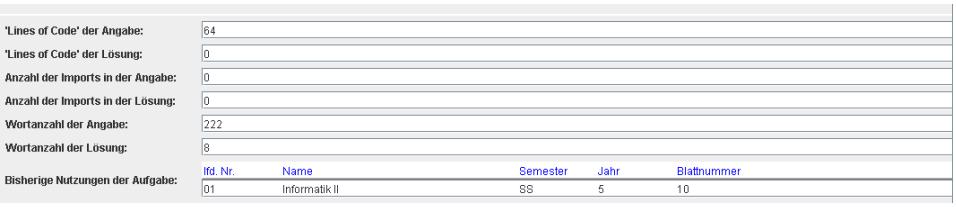
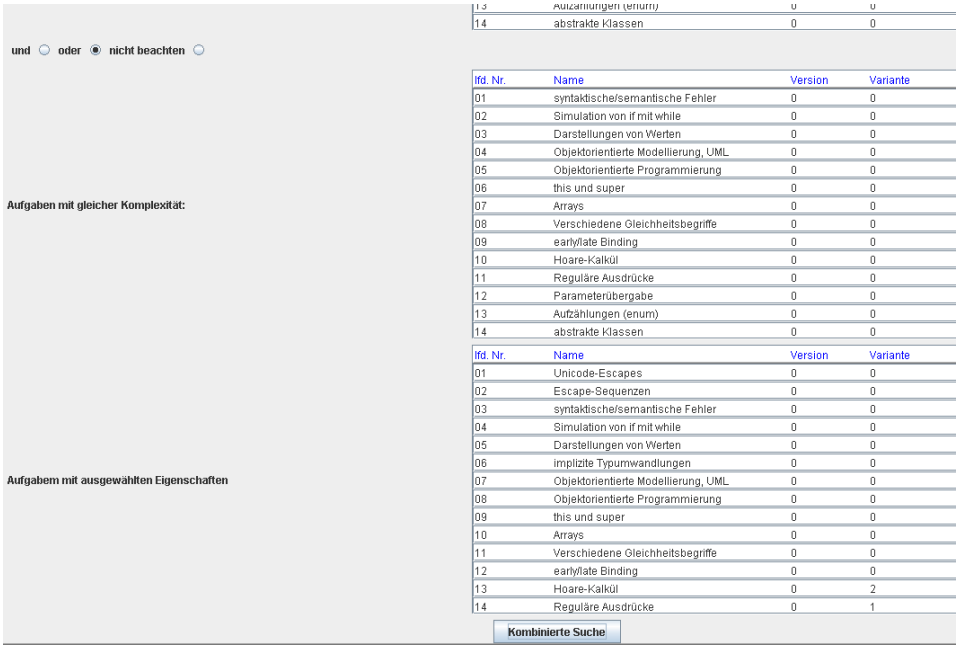


Abbildung 11: Numerische Informationen der Detailansicht

Ein letzter neuer Abschnitt gestattet es eine weitere Suchanfrage zu starten, jedoch nur mit den Suchkriterien die der Metaabschnitt bietet. Für jede einzelne Suchkriterium werden die Ergebnisse bereits angezeigt. Dies kann nützlich sein, wenn man im ersten Schritt relativ allgemein gesucht hat um einige Ergebnisse zu erhalten die im groben den geforderten Bedingungen entsprechen, dann aber noch eine bessere Einschränkung, z.B. nach Komplexität braucht. Es kann dazu des weiteren nach einer Kombination von Metainformationen gesucht werden. Die kombinierte Suche funktioniert dabei so, dass nur Kriterien betrachtet werden, deren Selektionsknopf auf entweder "und" oder "oder" steht, wobei dann die jeweilige logische Verknüpfung genutzt wird. Nutzt man diese Suchfunktion wird eine weitere Liste der Anzeige hinzugefügt, die das Ergebnis der kombinierten Suche darstellt. Wie bei SQL üblich, hat "und" dabei eine höhere Präzedenz als "oder". Geklammerte Ausdrücke werden nicht unterstützt. Als Beispiel sei hier der untere Abschnitt der Detailansicht gezeigt. In ihm ist exemplarisch nur die unterste Metainformation zu sehen. Der Verknüpfungsknopf zeigt eine Oder-Verknüpfung an.



lfd. Nr.	Name	Version	Variante
01	syntaktische/semantische Fehler	0	0
02	Simulation von if mit while	0	0
03	Darstellungen von Werten	0	0
04	Objektorientierte Modellierung, UML	0	0
05	Objektorientierte Programmierung	0	0
06	this und super	0	0
07	Arrays	0	0
08	Verschiedene Gleichheitsbegriffe	0	0
09	early/late Binding	0	0
10	Hoare-Kalkül	0	0
11	Reguläre Ausdrücke	0	0
12	Parameterübergabe	0	0
13	Aufzählungen (enum)	0	0
14	abstrakte Klassen	0	0

lfd. Nr.	Name	Version	Variante
01	Unicode-Escapes	0	0
02	Escape-Sequenzen	0	0
03	syntaktische/semantische Fehler	0	0
04	Simulation von if mit while	0	0
05	Darstellungen von Werten	0	0
06	implizite Typumwandlungen	0	0
07	Objektorientierte Modellierung, UML	0	0
08	Objektorientierte Programmierung	0	0
09	this und super	0	0
10	Arrays	0	0
11	Verschiedene Gleichheitsbegriffe	0	0
12	early/late Binding	0	0
13	Hoare-Kalkül	0	2
14	Reguläre Ausdrücke	0	1

Abbildung 12: Kombinierte Suche in der Detailansicht

Weiter aus der Detailansicht
Das weitere Vorgehen, nach dem Betrachten einer Detailansicht einer Übungsaufgabe, kann entweder sein, dass man den Tab mit "**Abbrechen**" schließt, dass man die **Aufgabe im Dateisystem erstellen** läßt, oder dass man eine neue Aufgabe davon ableitet bzw. die bestehende **Aufgabe bearbeitet**. Ableiten und Bearbeiten werden analog durchgeführt. Beim Erstellen einer Aufgabe wird das Verzeichnis, welches im zu wählenden Vorlesungsprofil spezifiziert ist, genutzt um die Program-Code Dateien, die Zusatzdateien, sowie den Latex-Quellcode der Aufgabe dort zu erstellen. Idee ist es dabei, eine Möglichkeit bereitzustellen, die Aufgabe in einer leslicheren Form zu präsentieren und extern zu nutzen um eine neue Aufgabe darauf aufzubauen. Das Ableiten bzw. Bearbeiten einer Übungsaufgabe wird nachfolgend im Abschnitt "**Aufgaben erstellen**" besprochen.

4.2.2 Aufgaben erstellen

Erste Möglichkeit
Aufgaben erstellt man in erster Linie durch drücken des entsprechenden Knopfes im linken Teil der Oberfläche. Man erhält daraufhin ein Formular, dass, bis auf die Version und Variantenangabe, mit dem Suchformular identisch ist. Wieder hat man die Möglicheit die Eingabe abzubrechen, indem man "**Abbrechen**" drückt. Will man jedoch eine Aufgabe tatsächlich abspeichern so drückt man "**In DB speichern**", nachdem man alle relevanten Informationen eingegeben hat. Die Applikation überprüft die Eingabe daraufhin. Ein Pop-Up mit den eventuell gefundenen Eingabefehlern erscheint wenn eines der folgenden Dinge gemacht wurde:

- Es wurde kein Titel angegeben.
- Es wurde eine der Metainformationen nicht angegeben.

- Der Angabentext ist leer.
- Der Angaben oder Lösungstext referenziert eine Datei die nicht in der Program-Code Liste bzw. Zusatzdateiliste enthalten ist.

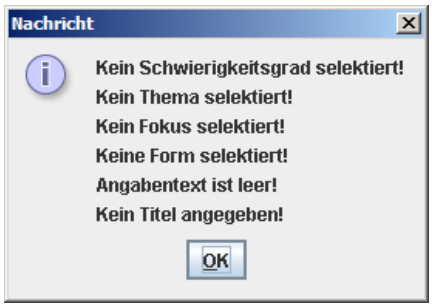


Abbildung 13: PopUp mit Information über fehlerhaftes Aufgabenanlegen

Wurde die Eingabe ohne Fehler validiert so wird sie in die Datenbank eingetragen, was durch ein entsprechendes Informations-Pop-Up bestätigt wird.

Abbildung 14: Aufgaben erstellen - Bestätigungs-PopUp

Es gibt jedoch noch einen Spezialfall. Wird als Aufgabenname ein bereits in der Datenbank vorhandener Aufgabenname gewählt reagiert die Applikation mit der Rückfrage ob man eine Version oder eine Variante anlegen möchte oder aber die bestehende Aufgabe abändern will.

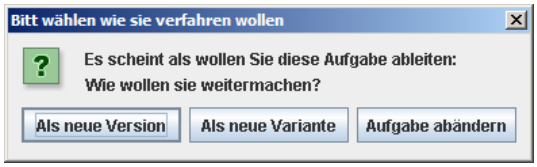


Abbildung 15: Aufgaben erstellen - Weiteres Vorgehen PopUp

Das liegt daran, das der Aufgabenname eindeutigt sein soll. Technisch gesehen ist das zwar nicht nötig und wird auch nicht so genutzt, allerdings kann man sonst die Aufgaben nicht als unterschiedlich visualisieren und ausserdem spiegelt ein gleicher Name auch wieder, dass die Aufgaben wohl sehr ähnlich, und somit eventuell einfach Varianten von einander, sind. Das Anlegen einer Variante oder Version wird nun im Zuge der zweiten Möglichkeit eine Aufgabe anzulegen, erläutert. Das Ändern einer bestehenden Aufgabe erfolgt ebenso wie in der zweiten Möglichkeit erklärt.

Zweite Möglichkeit
Ein anderer Weg eine Aufgabe anzulegen, besteht darin, sie von einer bereits bestehenden abzuleiten. Dies ist besonders sinnvoll, wenn man eine Variante einer Aufgabe anlegen will oder aber bei Änderungen an der Aufgabe, da auf diesem Wege weniger Aufwand entsteht. Ebenso ist es sinnvoll, wenn man eine neue Version einer Aufgabe anlegen will, also im Allgemeinen einen Fehler ausbessern will (zur Definition von Version und Variante siehe Kapitel "Entwurf", Abschnitt "inhaltlich"). Das Vorgehen ist dabei so, dass man sich, wie im vorigen Unterabschnitt beschrieben, zur Detailansicht der Aufgabe, die als Ableitungsbasis dienen soll, begiebt. Hier kann man nun den Knopf "**Aufgabe bearbeiten**" nutzen. Man bekommt eine neue Ansicht in der alle Eingabemöglichkeiten wieder aktiv sind, und die initial alle Daten der Basisaufgabe enthält. Der Aufgabenname wird zu "abgeleitet von:..." erweitert. Dies sollte so aber nicht übernommen werden. Des weiteren wird im Kommentar eine entsprechende Bemerkung hinzugefügt, da Versions- oder Variantenänderungen kommentiert werden sollten. Nachdem man nun die nötigen Anpassungen durchgeführt hat können die Formulardaten an die Datenbank geschickt werden. Wird die neue Aufgabe als Version oder Variante der Ableitungsbasis in der Datenbank abgelegt, wird dies mit einem PopUp quittiert. Zuvor wird jedoch noch geprüft ob sich überhaupt etwas geändert hat. Das jeweilige Ergebnis, also dass keine Änderung stattgefunden hat, eine Änderung fehlerhaft (siehe normales Anlegen) war oder dass der Datensatz erfolgreich gespeichert wurde, wird durch ein entsprechendes Pop-Up quittiert.
Die Aufgabe kann hingegegen auch mit verändertem Inhalt in gleicher Version und Variante abgespeichert werden. Dies kann zum Beispiel bei der Korrektur von syntaktischen Fehlern sinnvoll sein, um nicht unnötig hohe Versions- und Variantennummern zu erhalten. Es erscheint hierbei eine andere Dialogfolge. Die Applikation warnt den Nutzer, dass hierdurch bereits bestehende Übungsblätter mit beeinträchtigt werden (sofern sie dise Aufgabe enthalten). Dieser Mechanismus ist eine reine Vorsichtsmaßnahme um zu verhindern, dass bestehende Daten ungewollt verändert werden.

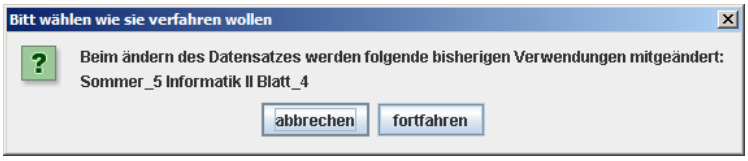


Abbildung 16: Aufgaben abändern - Weiteres Vorgehen PopUp

4.2.3 Übungsblatt erstellen

Die Komposition von Aufgaben zu einem Aufgabenblatt wird eingeleitet durch drücken des Knopfes "**Übungsblatt erstellen**" im linksseitigen Teil der Oberfläche. In der nun zu sehenden Oberfläche müssen ein paar Formulardaten eingetragen werden um ein Übungsblatt erstellen zu können. Als Erstes steht die Selektion eins Profils an. Dieses Profil wird genutzt um die zur Erstellung nötigen Parameter, wie Augabeordner und genutztes Latex-Package, einzulesen. Daraufhin muss angegeben werden welches Übungsblatt es sein soll, also welche laufende Nummer das Blatt bekommen soll. Im nächsten Abschnitt hat man die Möglichkeit eine Ankündigung auf das Übungsblatt zu setzen. Nun kommt der Kernteil dieses Formulars, die Aufgaben. Diese werden durch das betätigen des Knopfes "**Aufgabe hinzufügen**" zum Übungsblatt hinzugefügt. Letztlich hat man die Möglichkeit ein internes Feedback oder einen Kommentar anderer Art, der für interne Zwecke gedacht ist, dem Übungsblatt hinzuzufügen.

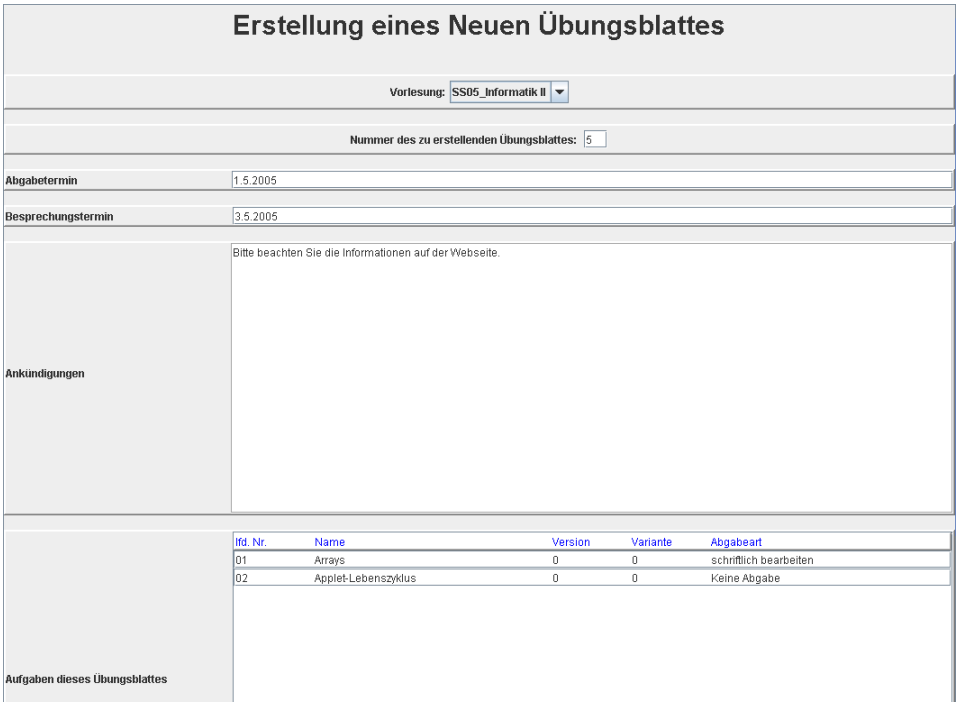


Abbildung 17: Übungsblatt erstellen

Hat man die nötigen Informationen eingetragen, so kann man, neben der stets verfügbaren Option den Tab zu schließen, das Aufgabenblatt entweder in die Datenbank eintragen, oder aber man kann das Aufgabenblatt im Verzeichnissystem, an der im Profil festgelegten Stelle, erstellen lassen. Die Erstellung des Aufgabenblattes läuft analog zur Erstellung einer einzelnen Aufgabe. Allerdings wird jeweils ein Ordner für die Dateien einer jeden Aufgabe erstellt und es gibt nur einen Latex-Quellcode der alle Aufgabenangaben- und lösungen enthält.Sowohl das Anlegen wie auch das Abspeichern werden, wenn keine Fehler in den Daten vorliegen, durch ein entsprechendes Pop-Up quittiert. Treten Fehler auf weden diese ebenso mittels eines Pop-Up mitgeteilt.

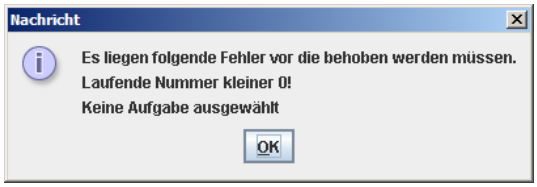


Abbildung 18: Fehler PopUp beim Übungsblatt erstellen

4.2.4 Übungsblatt betrachten

Bereits in der Applikation gespeicherte, alte Übungsblätter können durch drücken des Knopfes "**Übungsblatt betrachten**" angesehen werden. Dies ist nützlich um einen Vergleich zu früheren Vorlesungen haben zu können, und somit das gerade in Erstellung befindliche Übungsblatt (zum Beispiel im Sinne der Komplexität) besser bewerten zu können. Die Applikation erkundigt sich durch ein PopUp nach dem gewünschten Übungsblatt. Dieses wird

einer Oberfläche angezeigt, die analog zu der Oberfläche der Übungsblatt-Erstellung ist. Hier kann man nun das alte Übungsblatt im Dateisystem erstellen lassen um einen vollständigen Eindruck zu erhalten. Der Dialog via PopUp ist in der folgenden Abbildung zu sehen.

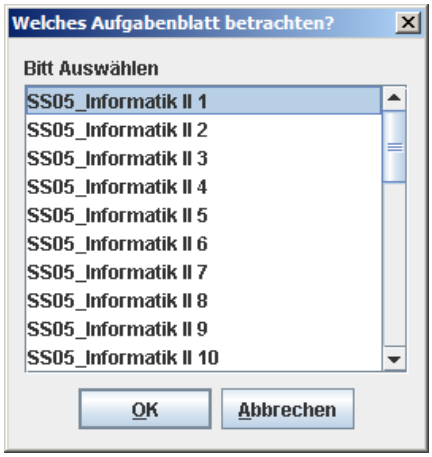


Abbildung 19: Welches Übungsblatt erstellen

4.2.5 Hilfsdatei In DB speichern

Will man dem System eine neue Hilfsdatei hinzufügen so drückt man den entsprechenden Knopf im linken Oberflächenteil. Daraufhin öffnet sich ein Dateiauswahl-Dialog in dem man eine beliebige Datei selektieren kann. Ist dies geschehen, ist noch anzugeben ob es sich um eine Programm-Code Datei oder eine Zusatzdatei handeln soll.

Abbildung 20: Hilfsdatei in DB speichern - Typauswahl

Der Unterschied besteht darin, dass Programm-Code Dateien mit Exkludierungen versehen werden können, Zusatzdateien im Gegensatz dazu, ungeachtet ihrer Form einfach byteweise ins System eingetragen werden. Hat man also diese Auswahl getroffen, erscheint eine "**Editor für Hilfsdateien**" genannte Oberfläche.

Diese erlaubt es, die in das System zu übernehmende Datei mit den nötigen Informationen anzureichern. Dazu verlangt die Oberfläche die Eingabe eines "Pseudonyms" für die Datei. Dies kann ein beliebiger Name sein, der die Datei gut beschreibt. Im Allgemeinen kann dies auch einfach der Dateiname noch einmal sein. Letzterer wird im nächsten Eingabefeld verlangt, ist aber schon voreingetragen mit dem Namen, den das Dateisystem liefert. Der Grund für diese Unterscheidung liegt in der Nutzung von Hilfsdateien in Übungsaufgaben. In vielen Fällen gibt es zu einer Aufgabe nur eine Hilfsdatei, die dann konsequent benannt werden kann. Um aber Dateien, die den selben Namen tragen müssen (in JAVA hat der Dateiname ja bekanntlich mit dem Typ zu tun), in der Darstellung (zum Beispiel in Listen) unterscheiden zu können und auch um den Inhalt durch einen Namen deutlicher

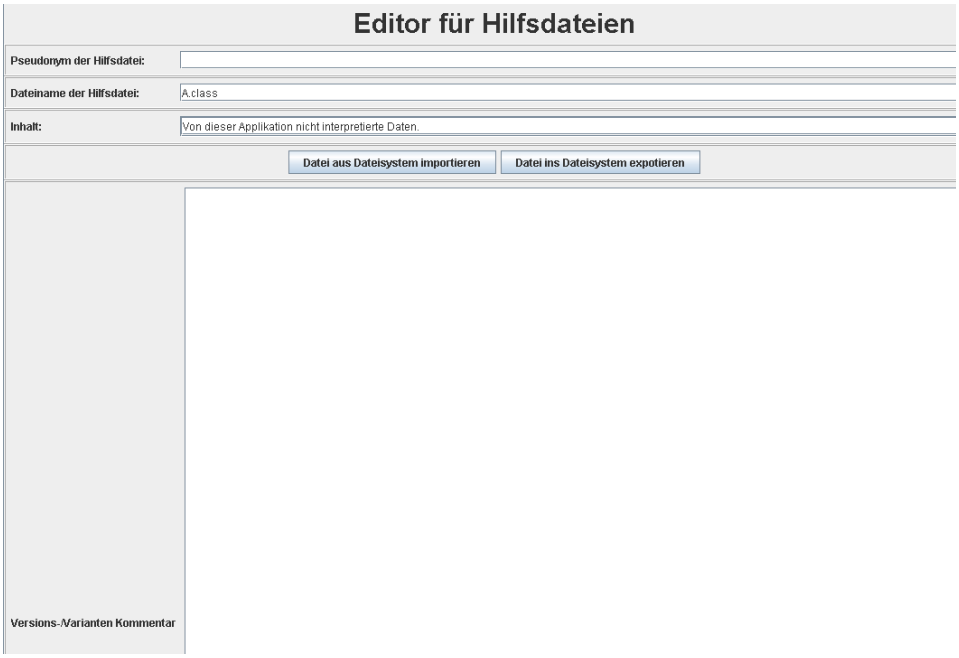


Abbildung 21: Hilfsdatei in DB speichern (hier: Zusatzdatei)

zu machen, wurde das Pseudonym eingeführt. Der nächste Teil variiert je nach Auswahl der Datei. Wird dem System eine Programm-Code Datei übergeben, so wird der Inhalt der Datei als Text interpretiert und in einem editierbaren Textfeld angezeigt. Handelt es sich jedoch um eine Zusatzdatei, so wird statt des Textfeldes nur die Mitteilung "Von dieser Applikation nicht interpretierte Daten." an dieser Stelle erscheinen. Ausnahme bilden hier ein paar gängige Grafikformate, die das System darstellen kann. Diese Grafiken werden dann nicht-editierbar dargestellt. Der Name "Editor" ist nicht so zu verstehen, dass eine komplette Editierumgebung angeboten werden soll. Vielmehr sind kleine Änderungen als Nutzungsszenario gedacht. Die eigentliche Editierung der jeweiligen Dateien soll mit dem Programm geschehen, dass den Präferenzen des Nutzers entspricht, da der Nutzer mit diesem sowieso besser umgehen kann. Letzlich kann man der Hilfsdatei noch einen **Versions-/Varianten Kommentar** hinzufügen. Primärer Verwendungszweck dieses Kommentarfeldes ist die Dokumentation von Änderungen bei neuen Versionen oder Varianten. Es spricht jedoch nichts dagegen auch andere, allgemeinere Informationen hier anzugeben, falls es angebracht scheint. Sollte man, nachdem man alle nötigen Informationen eingetragen hat, doch einen anderen Inhalt einsetzen wollen, so kann man durch betätigen des Knofpes "**Datei aus Dateisystem importieren**" nochmals einen Dateidialog öffnen. Diese Funktion macht allerdings beim Ändern einer Hilfsdatei mehr Sinn. Ebenso besteht die Möglichkeit den Inhalt wieder ins Dateisystem zurückzuschreiben. Dies ist mittels des Knopfes "**Datei ins Dateisystem exportieren**" möglich. Wiederum macht diese Funktion beim Ändern bereits bestehender Hilfsdateien mehr Sinn. Ist alles fertig eingetragen, so wird die Hilfsdatei durch drücken des Knopfes "**In DB speichern**" in die Datenbank übernommen. Vorausgesetzt natürlich, die Daten waren vollständig. Im positiven, wie im negativen Fall wird durch ein Pop-Up das Ergebnis mitgeteilt. Wie in jeder Oberfläche besteht auch die Möglichkeit mittels des "**Abbrechen**"-Knopfes die Oberfläche ohne Änderung zu verlassen.

4.2.6 Hilfsdatei ändern

Ähnlich wie bei Aufgaben, ist es auch bei Hilfsdateien möglich Varianten und Versionen anzulegen. Die primäre Verwendung ist sicherlich das Erstellen neuer Programm-Code Varianten um Aufgaben wiederverwendbar zu machen. Hierzu drückt man den entsprechenden Knopf in der linken Knopfleiste. Es erscheint eine Auswahlliste, die alphabetisch sortiert, erst alle Programm-Code Dateien und dann alle Zusatzdateien enthält. Hier wählt man nun die abzuleitende Datei aus. Ist dies geschehen wird eine Oberfläche, die mit der durch drücken des Knopfes "**Hilfsdatei hinzufügen**" erscheinenden identisch ist. Naheliegenderweise sind jedoch die Daten der selektierten Datei bereits eingetragen. Durch nutzen des Knopfes "**Datei aus Dateisystem importieren**" können die Daten geändert werden. Im Falle einer Programm-Code Datei können kleinere Änderungen auch im Textfeld selbst gemacht werden, wenn sich eine Bearbeitung in einem externen Editor nicht anbietet. Bestätigt man nun das Ableiten durch betätigen des Knopfes "**In DB speichern**" so erkundigt sich das System ob man eine neue Version oder eine neue Variante anlegen möchte. Wenn die Daten vollständig sind wird das jeweilige in die Datenbank eingetragen.

4.2.7 Vorlesungsprofil erstellen

Eine Vorlesung wird abstrakt charakterisiert durch ihr Profil. Um ein solches anzulegen drückt man den Knopf "**Vorlesungsprofil erstellen**". Daraufhin erscheint eine Oberfläche in der man ein Profil anlegen kann. Im obersten Abschnitt sind die bereits definierten Profile zu sehen, falls schon welche im System eingetragen sind. In den darunter liegenden Abschnitten erwartet das Formular die Eingabe der folgenden Informationen: Veranstaltungsjahr, Semester (unterschieden wird Winter- und Sommersemester), Veranstaltungstitel, das Standardausgabeverzeichnis sowie das zu nutzende Latex-Package. Die ersten Punkte sind selbsterklärend. Das Standardausgabeverzeichnis gibt an wo im Dateisystem Aufgaben bzw. ganze Übungsblätter abgelegt werden sollen, wenn sie erstellt werden. Das Latex-Package ist das zu Nutzende Package, welches die für die Vorlesung passenden Definitionen enthält. Durch drücken des "**Anlegen**"-Knopfes wird das Profil, falls es vollständig ist, in die Datenbank eingetragen. Wie bei den anderen Oberflächen kann man auch diese wieder durch drücken des "**Abbrechen**"-Knopfes verlassen. Die folgende Abbildung zeigt die unbearbeitete Maske, in der bereits drei andere Profile angelegt wurden.

4.2.8 Näheres zu einzelnen Komponenten

Hinzufügen von Programm-Code zu Aufgabe
Beim Hinzufügen von Programm-Code zu Aufgaben erkundigt sich das System nach zwei weiteren Informationen. Zum einen ist dies ob die Datei im Klartext oder Binär erscheinen soll. Im ersteren Fall wird der textuelle Inhalt ausgegeben, im zweiteren wird der Inhalt als Quellcode der Programmiersprache (JAVA) aufgefasst und kompiliert. Des weiteren erkundigt sich das System ob die Datei nur als Lösung oder für "beides" genutzt werden soll. Gemeint ist hier natürlich ob sich die Datei nur auf die Lösung bezieht oder ob sie auch den Studenten als Angabe dienen soll. Progamm-Code wird beim Eintragen in das System nach Exkludierungstags durchsucht. Dies sind zwei Tags (öffnender und schließender) die folgende Funktion haben: Jeglicher Programm-Code Text der zwischen diesen beiden Tags eingeschlossen ist, sowie die Tags selbst, werden aus der Studentenversion, der Datei entfernt, die Musterlösungsversion hingegen besteht aus dem gesamten Quellcode ohne die Exkludierungstags. Dies ist besonders hilfreich für die relativ häufig vorkommende Anforderung, einen Programmrahmen bereitstellen zu können, der von den Studenten fertig programmiert werden soll. Durch diese Methode ist eine einfachere Verwaltung möglich, da

Abbildung 22: Neues Profil anlegen

Änderungen nur noch an einer Stelle durchgeführt werden müssen.

Hinzufügen von Zusatzdatei zu Aufgaben
Das Hinzufügen von Zusatzdatei zu Aufgaben läuft analog zu dem Hinzufügen von Programm-Code zu Aufgaben, allerdings entfällt naheliegenderweise die Frage nach der Form (Binär oder im Klartext).

Selektion von Aufgaben über Listen
Listen, die Aufgaben enthalten, können für manche Zwecke durch die Darstellung von Namen und Versionen- bzw. Variantenzahl nicht genug Information bieten. Daher ist es bei Aufgabenlisten möglich, durch Doppelklicken die Detailansicht der so selektierten Aufgabe angezeigt zu bekommen.

Selektion von Programm-Code/ Hilfsdateien über Listen
Im gesamten System gilt: Ist in einer Liste ein Programm-Code oder eine Zusatzdatei angezeigt, so kann durch Doppelklicken der entsprechende Hilfsdateieditor geöffnet werden. Dies ist praktisch, um sich, den für Varianten und Versionen nützlichen, Kommentar anzusehen. Im Falle von Programm-Code Dateien jedoch sicherlich auch um auf einen Blick zu sehen um was es in dem Programm-Code im Einzelnen geht.

5 Import der bisherigen Aufgaben

5.1 Näheres zu den Aufgaben

Um die Entwicklung des Systems immer wieder mit den tatsächlich benötigten Funktionen abgleichen zu können, wurden die in den letzten Jahren verwendeten Übungsaufgaben in die Applikation integriert. Anfänglich wurden auch die Aufgaben des vorlesungsbegleitenden Buches des letzten Informatik 2 Semesters (2005) - "Java ist auch eine Insel" von Christian Ullenboom - analysiert um, deren Verwendbarkeit festzustellen. Sie erwiesen sich allerdings als weitestgehend nicht brauchbar. Im letzten Schritt der Entwicklung wurden alle Aufgaben des Vorlersungssemester Sommersemester 2001 und Sommersemester 2005 (jeweils gehalten von Prof. Dr. H.J. Ohlbach), sowie des Sommersemesters 2004 (gehalten von Prof. Dr. Martin Hofmann) in das System übernommen. Es handlet sich hierbei um insgesamt 117 Aufgaben die auf 33 Aufgabenblätter verteilt sind. Die Aufgaben nutzen insgesamt 209 Program-Code Dateien und 68 Zusatzdateien. Das Layout der Latex-Quellcode Dateien wurde dem im Sommersemester 2005 verwendeten Schema angepasst. Da die Formate zwar ähnlich, aber eben nicht gleich sind, können hier leicht kleinere Ungereimtheiten entstanden sein, die im Einzelnen erst beim Nutzen der Aufgabe ersichtlich werden dürften. Vorallem die geänderte Ordnerstruktur der beiden Veranstaltugnen des PMS Lehrstuhls, sowie die unterschiedliche Ordnerstruktur des TCS Lehrstuhls sollte im Gedächtnis behalten werden, wenn man Aufgaben übernimmt, da nicht immer alle nun verwendeten Informationen verfügbar waren.

5.2 Konventionen

Die bisherigen Aufgabe wurden unter Beachtung bestimmter Kriterien übernommen. Die drei Vorlesungsprofile wurden durch die Vorlesungsparamter bestimmt. Einzig das Ausgabeverzeichnis wurde auf ein lokales Verzeichnis zum Testen gesetzt. Das sollte man nach der Installation der Datenbank händisch ändern.

Die Abgabe- und Besprechungstermine der Übungsblätter wurden so wie sie waren übernommen. Ebenso wurde die Komposition der Aufgabenblätter nicht verändert. Insbesondere auch nicht die Frage, ob eine Aufgabe abzugeben ist oder nicht. Lediglich die Beschriftung wurde teils geändert. Aufgaben die auf den bisherigen Aufgabenblättern nicht abzugeben waren waren einfach ohne Kennzeichnung, nun haben sie jedoch den Kennzeichnung"Keine Abgabe" erhalten. Bei der TCS Vorlesung wurde generell kein Abgabevermerk gemacht. Die Aufgaben wurden daher willkürlich so ins System übertragen, als wäre jede Aufgabe abzugeben gewesen. Es wurden keinerlei Kommentare zu den Übungsblättern hinzugefügt, da hierfür keine Informationen vorliegen. Ankündigungen wurden, wenn auf einem Blatt vorhanden, eins zu eins übernommen.

Der Name einer Aufgabe wurde so wie er war übernommen. Die Bestimmung der Metainformationen geschah nach Bewertung des Inhalts im Vergleich zu den anderen Aufgaben. Die Programm-Code Dateien wurden um den Exkludierungsmechanismus angereichert, wo immer es Sinn machte. Die Einorndung in eine Aufgabenabfolge wurde, soweit möglich, durchgeführt. Themengebiete die Voraussetzung für eine Aufgabe sind wurden nur dann als solche gewertet wenn sie als "kürzlich erworbenes Wissen" zählen. Würde das nicht so sein, würde noch bei den komplexesten Aufgaben als abhängiges Thema zum Beispiel "Java Grundlagen" erscheinen, was an dem eigentlichen Ziel vorbei gehen würde. Welche Dateien zu der Angabe bzw. nur zur Lösung zählen, wurde anhand der Ordnerstruktur (im Falle der Vorlesungen Sommersemester 2004 und Sommersemester 2005) bzw. durch Interpretation der Aufgabe (im Falle der Vorlesung Sommersemester 2001) gemacht, je nachdem ob die Ordnerstruktur dafür geeignet war oder nicht. In einigen wenigen Fällen bot sich bereits die Gelegenheit eine Aufgabe als Variante einer anderen einzutragen. In diesen Fällen wurde

auch die “**Internes Feedback**“-Eingabe genutzt um das Delta zu beschreiben.

Zusatzdateien sowie Programmcode-Dateien wurden vollständig übernommen, wenn sie in der Aufgabe genutzt wurden. Andere Dateien, wie zum Beispiel weiterführende Dokumentationen, wurden ausgelassen. Die Programm-Code Dateien wurden mit den Exkludierungstags ergänzt, wo immer es sinnvoll war. Die Dateien erhielten ein Pseudonym das dem Programmnamen erweitert um den Text “JAVA“ entspricht. Dadurch wird klar, dass es sich um eine JAVA Datei handelt. Sollte die Vorlesung einmal eine andere objektorientierte Sprache benutzen ist diese Konvention angebracht. Programm-Code Dateien die nur “Main.java“ hießen, bekammen den Aufgabennamen, in der sie genutzt werden, als Präfix. Auch bei den Zusatzdateien und Programm-Code Dateien konnten bereits Variantenbeziehungen identifiziert werden, die dann auch so eingetragen wurden.

6 Implementierungsdetails für Weiterentwickler

6.1 Die Projektstruktur

Das gesamte Projekt ist übersichtlich in einzelne Ordner strukturiert. Im Projektordner selbst liegt die Applikation als Kompilat, in Form eines JAR-Archivs. Dieses ist ausführbar, es muss jedoch die weiter oben schon kurz angesprochene und unter **doc/README** beschriebene Vorgehensweise genutzt werden. Folgend soll in kurzer Form der Inhalt der einzelnen Projektordner beschrieben werden.

conf
Dieser Ordner enthält alle Konfigurationsdateien. Dazu zählen sowohl Konfigurationsdateien für die Applikation selbst, als auch solche die für die Weiterentwicklung des Systems nötig sind.

database
Dieser Ordner enthält alle persistenten Daten. Bei Fertigstellung der Projektarbeit ist dies ein Datenbankauszug in Standard-SQL, in Form einer Plain-Text-Datei.

doc
Dieser Ordner enthält die gesamte Projektdokumentation. Er ist weiter strukturiert. Im Ordner **Allgemeine Doku** sind Dokumente zu finden, die anfänglich zu den Implementierungsentscheidungen geführt haben und generell der dem Projekt eine Form gegeben haben. Sie sind für Weiterentwickler vielleicht interessant, um gewisse Designentscheidungen nachvollziehen zu können. Im Ordner **Ausarbeitung** ist dieses Dokument in verschiedenen Formaten zu finden, ebenso auch die in dieser Ausarbeitung verwendeten Abbildungen im eps-Format. Letztlich ist im Ordner **Javadoc** die automatisch aus dem Quelltext generierte Klassen- und Methodendokumentation zu finden.

lib
Dieser Ordner enthält alle benötigten Java-Bibliotheken. Bisher ist dies vor allem für die Datenbankanbindung nötig.

src
Dieser Ordner enthält den gesamten Quelltext der Applikation und wird weiter unten ausführlicher beschrieben.

6.2 Das Datenbankschema

Im folgenden soll das verwendete Datenbankschema dargestellt werden. Zu jeder Tabelle wird ein kurzer Einblick zu deren Verwendung gegeben. Dann wird die Funktion jeder Spalte geschildert. Letztendlich folgt eine tabellarische Darstellung.

6.2.1 answertext

Die Tabelle **answertext** enthält alle Texte die als Antwort einer Aufgabe angesehen werden. Also insbesondere im Allgemeinen nur für die Tutoren gedacht sind, außer die Lösung der Aufgabe soll explizit den Studenten zur Verfügung gestellt werden.

- **id**: Enthält die unique id der jeweiligen Aufgabenlösung.
- **assignment_id**: Enthält eine Refernz auf die Aufgabe zu welcher der Lösungstext gehört.
- **content**: Enthält die eigentlichen Nutzdaten, also die Aufgabenlösungen.

Feld	Typ	Null	Standard
id	int(10)	Nein	
assignment_id	int(10)	Nein	0
content	longtext	Ja	NULL

Tabelle 1: Struktur der Tabelle answertext

6.2.2 assignment

Die Tabelle **assignment** enthält alle notwendigen Daten einer Aufgabe an sich. Dies sind im folgenden:

- **id**: Enthält die unique id der jeweiligen Aufgabe
- **mode**: Enthält eine Referenz auf einen Eintrag in der Tabelle mode. Es wird hierdurch die Form der Aufgabe festgelegt (also zum Beispiel “Selbständig Programmieren“ oder “Fehler finden“).
- **topic**: Enthält eine Referenz auf einen Eintrag in der Tabelle topic. Es wird hierdurch das Thema der Aufgabe festgelget (also zum Beispiel “Threads“ oder “Generics“).
- **comment**: Enthält eine Referenz auf einen Eintrag in der Tabelle comment. Diese stellt das interne Feedback für diese Aufgabe dar.
- **focus**: Enthält eine Referenz auf einen Eintrag in der Tabelle focus. Es wird hierdurch der Fokus der Aufgabe festgelegt (also zum Beispiel “Java Konzepte“ oder “Algorithmik“).
- **complexity**: Enthält eine Referenz auf einen Eintrag in der Tabelle complexity. Es wird hierdurch die Komplexität einer Aufgabe festgelegt, wobei eine höhere Zahl für eine höhere Komplexität steht.
- **version**: Enthält die Version dieser Aufgabe. Eine Version ist, wie oben beschrieben, meist eine korrigierte Form einer Aufgabe und im Allgemeinen den älteren Versionen vorzuziehen.

- **variante**: Enthält die Variante dieser Aufgabe. Eine Variante ist, wie oben beschrieben, eine Abwandlung einer Aufgabe, die im Allgemeinen verwendet wird um Aufgaben wiederverwertbar zu machen.

Feld	Typ	Null	Standard
id	int(10)	Nein	
mode_id	int(10)	Nein	0
topic_id	int(10)	Nein	0
comment_id	int(10)	Ja	0
focus_id	int(10)	Nein	0
title	varchar(45)	Nein	
complexity_id	int(10)	Nein	0
version	int(11)	Nein	0
variante	int(11)	Nein	0

Tabelle 2: Struktur der Tabelle assignment

6.2.3 assignment_in_exercisesheet

Die Tabelle **assignment_in_exercisesheet** enspricht der Modellierung der Beziehung "Aufgabe x ist Bestandteil von Übungsblatt y". In ihr ist also festgehalten, aus welchen Aufgaben ein Übungsblatt besteht.

- **exerciseSheet_id**: Enthält eine Referenz auf das Übungsblatt, das die entsprechende Aufgabe enthält.
- **assignment_id**: Enthält eine Referenz auf die Aufgabe, die Bestandteil des Übungsblattes ist.
- **consecutive_number**: Enthält die laufende Nummer, also die Position, der Aufgabe auf dem Übungsblatt.
- **turnintype**: Enthält eine textuelle Beschreibung ob diese Aufgaben von den Studenten bearbeitet und abgegeben werden muss oder nicht.

Feld	Typ	Null	Standard
exerciseSheet_id	int(11)	Nein	0
assignment_id	int(10)	Nein	0
consecutive_number	int(11)	Nein	0
turnintype	varchar(45)	Ja	NULL

Tabelle 3: Struktur der Tabelle assignment_in_exercisesheet

6.2.4 assignments_published

Die Tabelle **assignments_published** enspricht der Modellierung der Beziehung "Aufgabenlösung x auf Blatt y in Vorlesung z veröffentlicht". In ihr ist also festgehalten, ob (wenn ein Eintrag besteht), und wann die Lösung einer Aufgabe bereits veröffentlicht wurde.

- **assignment_id**: Enthält eine Referenz auf die Aufgabe, deren Lösung veröffentlicht

wurde.

- **lecuter_id**: Enthält eine Referenz auf die Vorlesung (also welche und wann) in der die Lösung veröffentlicht wurde.
- **sheetnumber**: Enthält die Nummer des Übungsblattes auf dem die Lösung zu der Aufgabe veröffentlicht wurde.

Feld	Typ	Null	Standard
assignment_id	int(10)	Nein	0
lecture_id	int(10)	Nein	0
sheetnumber	int(10)	Nein	0

Tabelle 4: Struktur der Tabelle assignments_published

6.2.5 assignmenttext

Die Tabelle **assignmenttext** enthält alle Texte die als Angabe einer Aufgabe angesehen werden. Also im Allgemeinen der Text zwischen Aufgabentitel und Ende der Aufgabe.

- **id**: Enthält die unique id des jeweiligen Angabentexts.
- **assignment_id**: Enthält eine Refernz auf die Aufgabe zu welcher der Angabentext gehört.
- **content**: Enthält die eigentlichen Nutzdaten, also die textuelle Beschreibung was in einer Aufgabe zu tun ist.

Feld	Typ	Null	Standard
id	int(10)	Nein	
assignment_id	int(10)	Nein	0
content	longtext	Ja	NULL

Tabelle 5: Struktur der Tabelle assignmenttext

6.2.6 auxiliaryresrouce

Die Tabelle **auxiliaryresrouce** enthält alle nötigen Daten von Zusatzdateien, wie zum Beispiel Bildern oder HTML Seiten.

- **id**: Enthält die unique id der jeweiligen Zusatzdatei
- **content**: Enthält die Binärdaten der Zusatzdatei
- **name**: Enthält den Namen der Datei, so wie er für in der Applikation angezeigt wird
- **filename**: Enthält den Dateinamen der Zusatzdatei
- **version**: Enthält die Versionsnummer der Zusatzdatei
- **variante**: Enthält die Variantennummer der Zusatzdatei

- **comment**: Enthält eine Referenz auf einen Kommentar in der Tabelle **comment**, falls ein Kommentar überhaupt existiert.

Feld	Typ	Null	Standard
id	int(10)	Nein	
content	longblob	Ja	NULL
name	varchar(45)	Nein	
filename	varchar(45)	Ja	NULL
version	int(11)	Ja	NULL
variante	int(11)	Ja	NULL
comment	int(10)	Nein	0

Tabelle 6: Struktur der Tabelle auxiliaryresource

6.2.7 auxiliaryresrouce_in_assignment

Die Tabelle **auxiliaryresource_in_assignment** modelliert die Beziehung "Zusatzdatei x kommt in Aufgabe y vor".

- **auxiliaryresource_id**: Enthät eine Referenz auf eine Zusatzdatei.
- **assignment_id**: Enthält eine Referenz auf eine Aufgabe.

Feld	Typ	Null	Standard
auxiliaryresource_id	int(10)	Nein	0
assignment_id	int(10)	Nein	0
answerOnly	tinyint(1)	Nein	0

Tabelle 7: Struktur der Tabelle auxiliaryresource_in_assignment

6.2.8 comment

Die Tabelle **comment** enthält textuelle Anmerkungen jeglicher Art (also zum Beispiel internes Feedback)

- **id**: Enthält die unique id des Textes.
- **text**: Enthält den eigentlichen Text.

Feld	Typ	Null	Standard
id	int(10)	Nein	
text	text	Ja	NULL

Tabelle 8: Struktur der Tabelle comment

6.2.9 complexity

Die Tabelle **complexity** enthält eine Abbildung der Komplexität einer Aufgabe. In erster Instanz ist dies numerisch realisiert, wobei ansteigende Werte höhere Komplexität symboli-

sieren sollen.

- **id**: Enthält die unique id der dargestellten Komplexität.
- **value**: Enthält den (numerischen) Wert der dargestellten Komplexität.

Feld	Typ	Null	Standard
id	int(10)	Nein	
value	int(10)	Nein	0

Tabelle 9: Struktur der Tabelle complexity

6.2.10 exercisesheet

Die Tabelle **exercisesheet** enthält alle Informationen um ein komplettes Aufgabenblatt zu beschreiben.

- **id**: Enthält die unique id des Aufgabenblattes
- **lectures_id**: Enthält eine Referenz auf die Vorlesung in der dieses Übungsblatt gestellt wurde.
- **consecutive_Number**: Enthält die laufende Nummer dieses Aufgabenblattes innerhalb einer Vorlesung.
- **announcement**: Enthält eine Referenz auf eine Ankündigung, welche in der Tabelle **comment** gefunden werden kann.
- **comment**: Enthält eine Referenz auf ein internes Feedback, welches in der Tabelle **comment** gefunden werden kann.
- **turnindate**: Enthält eine textuelle Beschreibung, wann ein Aufgabenblatt abzugeben ist.
- **tutoringdate**: Enthält eine textuelle Beschreibung, wann ein Aufgabenblatt besprochen wird.

Feld	Typ	Null	Standard
id	int(11)	Nein	
lectures_id	int(11)	Nein	0
consecutive_Number	int(11)	Nein	0
announcement	int(10)	Nein	0
comment	int(10)	Nein	0
turnindate	varchar(60)	Nein	
tutoringdate	varchar(60)	Nein	

Tabelle 10: Struktur der Tabelle exercisesheet

6.2.11 focus

Die Tabelle **focus** enthält alle in der Applikation genuzten Foki. Sie gibt also an in welcher didaktischen Absicht die Aufgabe gestellt ist (zum Beispiel “Algorithmik“ oder “Objektorientierte

Konzepte“).

- **id**: Enthält die unique id des Fokuselements.
- **value**: Enthät den eigentlichen Fokuswert (also einen beschreibenden Text).

Feld	Typ	Null	Standard
id	int(10)	Nein	
value	varchar(45)	Ja	NULL

Tabelle 11: Struktur der Tabelle focus

6.2.12 lecture

Die Tabelle **lecture** enthält alle im System gespeicherten Vorlesungsprofile und deren Vorlesungsattribute.

- **id**: Enthält die unique id des Vorlesungsprofils.
- **term**: Enthält eine Beschreibung des Semesters (also im Allgemeinen Winter- bzw. Sommersemester).
- **title**: Enthält den Namen der Vorlesung (also zum Beispiel “Informatik II“).
- **year**: Enthält das Jahr, in dem die Vorlesung stattgefunden hat.
- **outputdir**: Enthält das Verzeichnis, in welchem Übungsblätter erstellt werden sollen.
- **latexpackage**: Enthält das für diese Vorlesung benötigte Latex-Package

Feld	Typ	Null	Standard
id	int(11)	Nein	
term	varchar(45)	Nein	
title	varchar(255)	Nein	
year	tinyint(4)	Nein	0
outputdir	varchar(60)	Nein	
latexpackage	varchar(45)	Nein	

Tabelle 12: Struktur der Tabelle lecture

6.2.13 mode

Die Tabelle **mode** enthält alle bekannten Bearbeitungsmodi (also zum Beispiel “Eigenständig Programmieren“ oder “Multiple Choice“).

- **id**: Enthält die unique id des Bearbeitungsmodus.
- **value**: Enthät den eigentlichen Bearbeitungsmodus (also einen beschreibenden Text).

Feld	Typ	Null	Standard
id	int(10)	Nein	
value	varchar(45)	Ja	NULL

Tabelle 13: Struktur der Tabelle mode

6.2.14 needs_prior_knowledge_from

Die Tabelle **needs_prior_knowledge_from** modelliert die Beziehung "Aufgabe x benötigt Vorwissen aus Themengebiet y". Für ein Element dieser Beziehung bedeutet dies also, dass das enthaltene Themengebiet inhaltliche Voraussetzung für die sinnvolle Bearbeitung der Aufgabe ist.

- **topic_id**: Enthält eine Referenz auf das Thema, welches als thematische Voraussetzung gelten soll.
- **assignment_id**: Enthät eine Referenz auf die Aufgabe, welche eine thematische Voraussetzung hat.

Feld	Typ	Null	Standard
topic_id	int(10)	Nein	0
assignment_id	int(10)	Nein	0

Tabelle 14: Struktur der Tabelle needs_prior_knowledge_from

6.2.15 order

Die Tabelle **order** modelliert die Beziehung "Aufgabe x ist Vorgänger von Aufgabe y". Es wird also dargestellt, dass eine gewisse Aufgabe im Allgemeinen nur sinnvoll ist, wenn die notwendige Vorgängeraufgabe bereits gestellt wurde.

- **Predecessor**: Enthält eine Referenz auf die Aufgabe, welche die Vorgängeraufgabe dieses Tupels darstellt.
- **Successor**: Enthät eine Referenz auf die Aufgabe, welche die Nachfolgeraufgabe dieses Tupels darstellt.

Feld	Typ	Null	Standard
Predecessor	int(10)	Nein	0
Successor	int(10)	Nein	0

Tabelle 15: Struktur der Tabelle order

6.2.16 person

Die Tabelle **person** wird zur Zeit noch nicht genutzt. Sie ist gedacht um Vorlesungsverantwortliche ebenso in dem System zu pflegen, um diese gegebenenfalls dann automatisch in den Latexkopfteil zu übernehmen.

- **id**: Enthält die unique id der Person.

- **lectures_id**: Enthält eine Referenz auf die Veranstaltung, in der die Person als Verantwortlicher tätig ist.
- **name**: Enthält den Namen der Person.

Feld	Typ	Null	Standard
id	int(10)	Nein	
lectures_id	int(10)	Nein	0
name	varchar(45)	Ja	NULL

Tabelle 16: Struktur der Tabelle person

6.2.17 programcode

Die Tabelle **programcode** enthält alle Hilfsdateien die ausführbaren Programmcode, der in der Vorlesung verwendeten Programmiersprache (bis auf weiteres wohl JAVA), enthalten. Die Tabelle ist ähnlich zur Tabelle **auxiliaryresource**, ist aber eigentständig, da in der Applikation eine abweichende Behandlung der beiden Hilfsdateitypen erfolgt.

- **id**: Enthält die unique id des Programmcodes.
- **content**: Enthält die eigentlichen Nutzdaten, also den eigentlichen Programmcode.
- **name**: Enthält den Namen des Programmcodes so wie er für den Nutzer (zum besseren Verständnis) in der Applikation erscheint.
- **filename**: Enthält den Dateinamen des Programmcodes, der bei der Erstellung der Aufgabe genutzt wird.
- **version**: Enthält die Versionsnummer des Programmcodes.
- **variante**: Enthält die Variantennummer des Programmcodes.
- **comment**: Enthält eine Referenz auf die Tabelle **comment**. Diser stellt einen internen Kommentar zu der Aufgabe dar, der zum Beispiel als Feedbackmöglichkeit genutzt werden kann oder Versions-/Variantendeltas enthält..

Feld	Typ	Null	Standard
id	int(10)	Nein	
content	longtext	Ja	NULL
name	varchar(45)	Nein	
filename	varchar(45)	Ja	NULL
version	int(11)	Ja	NULL
variante	int(11)	Ja	NULL
comment	int(10)	Nein	0

Tabelle 17: Struktur der Tabelle programcode

6.2.18 programcode_in_assignment

Die Tabelle **programcode_in_assignment** modelliert die Beziehung "Programmcode x gehört zu Aufgabe y". In ihr steht also, welche Programmcodes zu welcher Aufgabe gehören, als auch in welcher Form der Programmcode in der Aufgabe auftritt.

- **programcode_id**: Enthält eine Referenz auf den Programmcode, der in der Aufgabe verwendet wird.
- **assignment_id**: Enthält eine Referenz auf die Aufgabe, die den Programmcode verwendet.
- **binaryVersion**: Enthält einen Wahrheitswert (bzw. 0/1 in SQL) der angiebt, ob der Programmcode in der Aufgabe in kompilierter Version oder in Klartext vorliegt.
- **hasExclusions**: Enthält einen Wahrheitswert (bzw. 0/1 in SQL) der andeutet, ob der Programmcode in dieser Aufgabe mit Exkludierungstags versehen ist, oder nicht.
- **answerOnly**: Enthält einen Wahrheitswert, der angiebt, ob der Programmcode nur zur Lösung zählt, oder auch mit der Angabe den Studenten zur Verfügung gestellt werden soll.

Feld	Typ	Null	Standard
programcode_id	int(10)	Nein	0
assignment_id	int(10)	Nein	0
binaryVersion	tinyint(1)	Nein	0
hasExclusions	tinyint(1)	Nein	0
answerOnly	int(10)	Nein	0

Tabelle 18: Struktur der Tabelle programcode_in_assignment

6.2.19 topic

Die Tabelle **topic** enthält alle bekannten Themengebiete (also zum Beispiel "Threads" oder "Imperatives Programmieren").

- **id**: Enthält die unique id des Themengebiets.
- **value**: Enthält das eigentliche Thema (also eine textuelle Beschreibung).

Feld	Typ	Null	Standard
id	int(10)	Nein	
value	varchar(45)	Ja	NULL

Tabelle 19: Struktur der Tabelle topic

6.2.20 topic_subtopic

Die Tabelle **topic_subtopic** modelliert die Beziehung "Themengebiet x ist Vaterelement von Themengebiet y". Die Tabelle bildet zwischen Themengebieten eine hierarchische Beziehung ab oder anders gesagt: in dieser Relation wird definiert, was übergeornetes Themengebiet von einem anderen Themengebiet ist.

- **inferior**: Enthält eine Referenz auf ein Themengebiet, welches als Unterthema modelliert wird.
- **superior**: Enthält eine Referenz auf ein Themengebiet, welches als übergeordnetes Themengebiet modelliert ist.

Feld	Typ	Null	Standard
inferior	int(10)	Nein	0
superior	int(10)	Nein	0

Tabelle 20: Struktur der Tabelle topic_subtopic

6.3 Der Quellcode

In diesem Kapitel soll ein kurzer Überblick über die Struktur des Quellcodes gegeben werden. Da extensiv Javadoc genutzt wurde, entfällt in dieser Ausarbeitung eine Erklärung in einer all zu großen Detailtiefe. Vielmehr soll die Grobstruktur und einzelne, vielleicht nicht auf Anhieb ersichtlichte, Konzepte erläutert werden.
Generell befolgt die Quellcodestruktur offizielle Konventionen. Alle Packages mit Quelltexten liegen in folgendem übergeordneten Package: **de.lmu.ifi.pms.aufsam** (wobei aufsam der interne Entwicklungsname ist). Der gesamte Quelltext ist weiterhin in einige übergeordnete Packages strukturiert. Folgend wird Package für Package dessen Inhalt kurz angerissen.

6.3.1 controller

Da von vornherein daran gedacht wurde, die Applikation später eventuell auf ein webbasiertes System zu portieren, wurde das dafür gut geeignete MVC (Model-View-Controller) Pattern gewählt. Alle Bestandteile des Ornders **controller** enthalten daher die Business Logik-steuernden Komponenten. Der Ordner enthält für jeden logischen Baustein (also jeden prinzipiell unterschiedlichen Prozess) eine Quellcodekomponente, um eine möglichst gute Modularität zu gewährleisten.

6.3.2 database

In diesem Package sind generell alle Quellcodekomponenten beheimatet, die mit einer persistenten Datenspeicherung beauftragt sind. Im Moment ist dies nur eine einzige Quelltextdatei die für einen Datenbankzugriff mittels Standard-SQL sorgt.

6.3.3 exceptions

In diesem Package sind global alle möglichen Ausnahmebehandlungen der gesamten Applikation zu finden.

6.3.4 model

Dieses Package stellt die Model-Komponente des oben erwähnten MVC-Patterns dar. In ihm sind also alle applikationsweiten Datenrepräsentationen zu finden. Die interne Struktur der einzelnen Komponenten ist jeweils auf ein Applikationsobjekt (also zum Beispiel Modellierung eines Vorlesungsprofils) ausgelegt. Ausnahme davon bilden die verschiedenen Aufgabenrepräsentationen, welche in einer Vererbungshierachie stehen. Grund hierfür war die Minimierung der zu übertragenden Daten von der Datenbank aus Performanzgründen.

6.3.5 utilities

In diesem Package sind verschiedene nützliche Quelltextkomponenten zu finden, die in der Applikation genutzt werden, aber so allgemein sind, dass eine eigene Struktur unangebracht ist, um die möglicherweise hilfreiche Wiederverwendung zu erleichtern.

6.3.6 validator

In diesem Package sind Komponenten zu finden, die die Datenintegrität sicher stellen indem sie die Eingabe des Nutzers auf Korrektheit überprüfen.

6.3.7 view

Dieses Package stellt die View-Komponente des oben erwähnten MVC-Patterns dar. In ihr ist also die Darstellung der Applikation beheimatet. Generell ist, analog zur Controller-Komponente, jeweils eine Quelltextdatei pro logischem Prozess zu finden. Das Package selbst enthält keinen Quelltext. Stattdessen enthält das Package zum einen das Package **interfaces**, welches die Eigenschaften der einzelnen Darstellungskomponenten enthält. Zum anderen sind hier die Packages der jeweiligen Darstellungskomponenten beheimatet. Bisher implementiert ist eine Swing-basierte View, welche im Package **swing** zu finden ist. Das Package **swing** wiederum enthält die Implementierung der vorher genannten Interfaces, also mehrere Quelltextdateien (jeweils eine pro logischem Prozess). Des weiteren ist in diesem Package ein weiteres Package namens **utilities** zu finden. In diesem sind nützliche Grafikkomponenten zu finden, die genutzt werden, um die vorhandene Swing-Darstellung zu ermöglichen, die aber auch in anderen Swing-basierten Implementierungen der View-Komponente genutzt werden können.

7 Weiterer Ausblick

Schon bei der Anforderungsdefinition war sowohl den Aufgabenstellern, als auch dem Bearbeiter klar, dass es unwahrscheinlich ist, dass die Applikation in der vorliegenden Form vollkommen an die Ansprüche angepasst sein würde. Es wird wohl erst ein oder vielleicht auch mehrere Semester brauchen, um festzustellen welche Applikationseigenschaften im Alltagsgeschäft wirklich benötigt werden, welche fehlen und welche, so wie sie sind, noch unzureichend sind. Aus diesem Wissen heraus wurde versucht die Applikation so gut wie möglich modular und damit erweiterbar und veränderbar zu halten. Obwohl die wirklich nötigen Änderungen erst durch Verwendung des Systems klarer werden, gibt es vom Applikationsersteller ein paar Punkte die mögliche, sinnvolle Erweiterungen darstellen können. Diese sollen hier in kurzer Form als Anregung für Weiterentwickler dargestellt werden. Die ersten sechs Vorschläge sind eher inhaltlicher Natur, die restlichen sieben eher technischer Natur.

7.1 Mehrere Themengebiete für eine Aufgabe

Bisher selten oder gar nicht festzustellen war der Fall, dass eine Aufgabe durchaus für zwei unterschiedliche Lehrzwecke genutzt werden kann. In diesem Fall müsste die Aufgabe auch zwei Themengebieten zuordenbar sein. Dies ist mit der bisherigen Version nicht möglich. Vielleicht entsteht jedoch in Zukuft Bedarf für diese Möglichkeit. Technisch gesehen ist die Erweiterung relativ unkompliziert und wird wohl hauptsächlich die übersichtliche Darstellungsweise betreffen.

7.1.1 Mehrere Bearbeitungsmodi für eine Aufgabe

Eine ähnliche Idee wie bei vorigem Punkt liegt dieser Erweiterung zu Grunde. Es könnte sinnvoll werden, mehrere Bearbeitungsmodi für eine Aufgabe angeben zu können. Dies ist gerade bei Aufgaben mit Unteraufgaben denkbar.

7.1.2 JAVA Version miteinbeziehen

Bisher wird in den Metadaten nicht gespeichert ab welcher JAVA Version ein Quelltext kompilierbar ist. Trotz des Umstiegs auf JAVA 5.0 hat sich dafür kein zwingender Grund ergeben. Vielleicht ist es jedoch in Zukunft interessant.

7.1.3 Mehrere Lösungen für eine Aufgabe

Hierunter ist zu verstehen, dass eine Aufgabe verschiedene, äquivalente Lösungen haben kann, die alternative Lösungswege darstellen können. Es hat sich allerdings bisher nicht die Notwendikeit gezeigt, da ein Editieren der einizigen Lösung damit äquivalent ist. Sollte es in Zukunft jedoch häufiger vorkommen, dass es verschiedene Lösungswege gibt, wäre eine Aufspaltung in mehrere Lösungen, der Übersichtlichkeit wegen, lohnenswert.

7.1.4 Validieren der Inkludierung

Es werden in der Applikation schon einige Validierung durchgeführt. Zum Beispiel verweigert die Applikation das Speichern einer Aufgabe, wenn im Latex-Quellcode eine Hilfsdatei referenziert wird, die nicht in einer der Hilfsdateilisten angegeben ist. Es könnte sich, als Erweiterung hierzu, als nützlich erweisen, die import-Beziehungen eingebundener Dateien ebenso zu validieren. Wenn also eine Aufgabe eine JAVA Quelltext Datei enthält die ihrerseits aber eine zweite JAVA Quelltext Datei zum korrekten Kompilieren benötigt, so sollte die Applikation auf das Vorhandensein dieser zweiten JAVA Quelltext Datei achten und gegebenenfalls eine Warnung geben, falls die Datei fehlt. Man könnte hierzu die import-Anweisungen nutzen.

7.1.5 Exkludierungstags erweitern

Bisher kann in einer Datei nur eine (Sequenz von) Exkludierungen auftreten, die ganz oder gar nicht durchgeführt werden. Manche Aufgaben können jedoch, je nach Ausprägung der Aufgabe, verschiedene Exkludierungen benötigen. So könnte zum Beispiel eine Programm-Code Datei, wenn in Aufgabe *A* vorkommend, nur die Methode *foo()* exkludiert haben, während sie, in Aufgabe *B* vorkommend, nur die Methode *bar()* exkludiert hat.

7.2 Implementierung des Datenbankzugriffs

Gemeint ist hier die Erweiterung der bisherigen Implementierung. Diese ist voll funktionsfähig, fehlergetestet und arbeitet wie sie soll. An manchen Stellen ist die Performanz aber nicht optimal. Während der Entwicklung stellte dies keinen Flaschenhals dar und war daher kein Problem. Aller Wahrscheinlichkeit nach werden die Datenmengen nie eine so kritische Menge erreichen, als dass ein wirkliches Performanzproblem entstehen würde. Allerdings ist es ein Punkt der zumindest nicht vergessen werden sollte, und der vielleicht bei einer Überarbeitung der Applikation mit bearbeitet werden kann.

7.3 Unterschiedliche Datenhaltung

Da für das primäre Nutzungsszenario sich aus gegebenen Gründen eine persistente Speicherung der Daten in einer Datenbank anbot und eine Änderung als nicht so dringlich erschien, wurde auch nur eine Datenbankschnittstelle implementiert. Eventuell könnte es jedoch reizvoll sein, die Daten auch in anderer Form (zum Beispiel im XML-Format) zu speichern. In diesem Fall ist die Datenhaltungskomponente im Quelltext noch zu erweitern. Die Notwendigkeit hierfür ist jedoch wohl eher akademischer und nicht so sehr praktischer Natur. Wahrscheinlicher könnte eine Festlegung auf eine bestimmte Datenbank aus Performanzgründen sein (siehe oberen Punkt). Damit einher geht dann die Ungenügsamkeit des bisherigen Datenbankzugriffs der nur Standard-SQL enthält.

7.4 Webbasierter Zugriff

Die bisherige Implementierung setzt auf die Darstellung mittels Swing, und somit also auf die Nutzung als eine Java-Applikation. Es könnte durchaus sinnvoll sein die Applikation auf eine webbasierte Form zu portieren um den einfacheren Zugriff darauf zu ermöglichen. Es stellt sich also die Herausforderung, eine J2EE-Web-Applikation zu erstellen, die das bisherige System nutzen kann. Der Quellcode ist so beschaffen, dass abgesehen von der Infrastruktur die man einrichten muss, lediglich eine View-Komponente erstellt weden muss, die mittels Servlets gelenkt wird.

7.5 Schnittstellen nach aussen

Bis zur Fertigstellung des Projekts wurde von einem System ausgegangen, dass ausschließlich zur Erstellung, Archivierung und Wartung von Übungsaufgaben genutzt werden soll. Es ist jedoch durchaus denkbar und auch sinnvoll, das System dahingehend zu erweiteren, dass es mit weiteren Komponenten zu einem ausgereiftem Hilfswerkzeug für eine gesamte Vorlesung gemacht werden kann. Dafür ist die sinnvolle Definition von Service-Schnittstellen von Nöten. Da jedoch diese Erweiterung bisher ein reines Gedankenkonstrukt ist, und keine speziellen weiteren Komponenten angedacht sind, war die Schnittstellendefinition bisher nicht in sinnvollem Maße möglich und wurde daher unterlassen.

7.6 Import von neuen Aufgaben

Die Applikation enthält im Zustand der Projektabgabe, den Aufgabenfundus von zwei Informatik 2 Vorlesungen des PMS-Lehrstuhls, sowie eine des TCS Lehrstuhls. Es ist denkbar, dass man von anderen Lehrstühlen weitere Aufgaben in das System aufnehemen kann, oder aber aus Büchern übernehmen kann. Das Einpflegen der neuen Daten in das vorhandene System ist unter diesem Punkt gemeint.

7.7 Erweiterung der Unterstützung für weitere Quelltextsprachen

Die Applikation ist so weit möglich darauf ausgelegt unabgängig von einer bestimmten Programmiersprache zu sein. Es gibt allerdings Stellen die dennoch abhängig von der Programmiersprache sind. Ein Beispiel dafür ist die Generierung der Ordnerstruktur durch Nutzung der JAVA Package Information. In anderen Programmiersprachen ist dies nicht immer so und bedarf dann einer anderen Behandlung.

7.8 Erweiterung der Unterstützung für andere Vorlesungen

Die Applikation ist prinzipiell nicht auf eine bestimmte Vorlesung begrenzt, allerdings wurden die Bedürfnisse für eine Informatik 2 Vorlesung als Grundlage genutzt. Es ist daher durchaus möglich, dass bei anderen Vorlesungen andere Anforderungen aufkommen können. Dies gilt es zu evaluieren und gegebenenfalls zu erweiteren.

Abbildungsverzeichnis

Tabellenverzeichnis