Technische Universität Dresden, Lehrstuhl Rechnernetze

großer Beleg

Untersuchung von Datenbasen und Match-Making im Peer Assessment

Josephine Seifert

25. November 2016

Betreut durch: Dr.-Ing. Tenshi Hara
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Großer Beleg im Arbeitskreis Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill an der Technischen Universität Dresden

Arbeit erstellt mit LuaLaTeX

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur Lehrstuhl Rechnernetze

AUFGABENSTELLUNG FÜR DIE BELEGARBEIT

THEMA: Untersuchung von Datenbasen und Match-Making im Peer Assessment

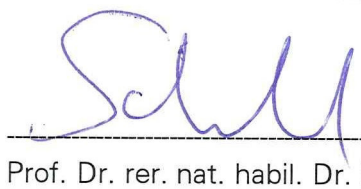
Name, Vorname:	Seifert, Josephine	Studiengang:	Dipl.-Inf. (PO 2009)
Matrikel-Nummer:	3578603	Projekt/Schwerpunkt:	Tech-enhanced Learning
Verantw. HSL:	Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill		
Involvierte Mitarbeiter:	Dipl.-Inf. Tenshi Hara, Dr.-Ing. Iris Braun		
Beginn am:	26.05.2016	Einzureichen bis:	25.11.2016

ZIELSTELLUNG

In der Regel ist für ein erfolgreiches universitäres Lernen in regelmäßigen Abständen eine Überprüfung des Lernfortschritts notwendig, beispielsweise durch Prüfungen. Im Sinne des selbstregulierten Lernens wird die Überprüfung umso wichtiger. In der Praxis ist jedoch oftmals aus Zeitgründen nur eine Prüfung am Ende des Semesters vorgesehen, um den Workload der involvierten Mitarbeiter gering zu halten. Dies hat im Umkehrschluss den negativen Effekt mangelnder Leistungsstandrückmeldung an die Studierenden.
Ziel dieser Belegarbeit ist daher die Untersuchung von Möglichkeiten, den Studierenden regelmäßige Leistungsstandrückmeldungen zu geben und gleichzeitig den Workload der involvierten Mitarbeiter zu senken. Dabei soll vor allem auf die empfohlenen Rollenwechsel Rücksicht genommen werden. Das bedeutet, Studierende sollen in die Rolle des Lehrenden und Prüfenden versetzt werden, um ihr eigenes Wissen besser verstetigen zu können. Ein diesen Kriterien zuträglicher Ansatz ist das Peer Assessment, bei dem Studierende sich gegenseitig bewerten. Die Problematik dabei ist jedoch, wie das Match-Making für das Peer Assessment stattfindet. Hier sind beispielsweise Ranglisten denkbar, welche durch geeignete Gamification Methoden befüllt werden.
Die Untersuchungen sollen in einer Bewertung des Ist-Standes und einer Empfehlung münden. Dazu ist die Befragung von Probanden und/oder die Implementierung eines Demonstrators denkbar.

SCHWERPUNKTE

- Recherche verwandter Arbeiten und Lösungen mit Schwerpunkt auf Peer Assessment,
- Use-Case-Analyse mit den möglichen, sinnvollen, universitären Szenarien,
- Diskussion verschiedener Lösungen,
- ggf. prototypische Implementierung und Probandenbefragungen, und
- Evaluation und Bewertung der Ergebnisse.



Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
(verantwortlicher Hochschullehrer)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation und Zielstellung

Die Wissensvermittlung an Universitäten erfolgt zum Großteil durch Vorlesungen. Das so erworbene Wissen wird am Ende des Semesters durch eine Prüfungsleistung evaluiert. Teilweise sind mehrere Vorlesungen, die auch semesterübergreifend angeboten werden können, Bestandteil einer solchen Prüfung und die Studenten bekommen erst aufgrund dessen eine Rückmeldung über ihren Leistungsstand. Während des gesamten Zeitraums des Lernens und der Vorbereitung auf eine Prüfung erfolgt hingegen keine Überprüfung des Lernfortschritts.

Um den Studenten trotzdem die Möglichkeit einer fortwährenden und die Lehrveranstaltung begleitende Überprüfung ihres eigenen Leistungsstands zu geben, kann Peer Assessment genutzt werden. Somit können sich die Studenten selbst gegenseitig bewerten und helfen. Dadurch werden den involvierten Mitarbeitern der Lehrveranstaltungen keine zusätzlichen Belastungen zugemutet und die Studenten können sich in die Rolle von Lehrenden versetzen. Dies ist auch der Verfestigung von Wissen dienlich.

In dieser Arbeit soll die Problematik des Match-Making für ein solches Peer Assessment behandelt werden. Dafür werden zunächst in Kapitel 2 verschiedene schon vorhandene Algorithmen und Protokolle für Match-Making und ihre Tauglichkeit für diesen Anwendungsbereich diskutiert. Zusätzlich erfolgt ein Einblick in den Bereich Gamification, um auf die Aspekte der Motivation der Studenten einzugehen, die ein solches Angebot für Peer Assessment nutzen sollen. Anschließend werden in Kapitel 3 Anwendungsfälle im universitären Alltag untersucht, in dem die Studenten von Peer Assessment profitieren können. Dabei wird auf verschiedene Möglichkeiten eines Match-Makings eingegangen und die Voraussetzungen dafür diskutiert. Danach soll eine Befragung der Zielgruppe klären, in wie weit so ein Programm angenommen und genutzt wird und welche Funktionen sich die Studenten wünschen würden. Aufbauend auf diesen Grundlagen soll in Kapitel 4 ein Ansatz für das Match-Making zwischen Studenten konzipiert werden. Zudem soll dieser Ansatz durch eine weitere Befragung von Studenten in seiner Anwendbarkeit evaluiert und verbessert werden. Diese Konzeption soll schließlich prototypisch implementiert und diese Implementation in Kapitel 5 vorgestellt werden. Abschließend werden die Ergebnisse ausgewertet und eine Empfehlung zum Peer Assessment im universitären Bereich, sowie ein Ausblick auf weitere Arbeiten zu diesem Themenbereich gegeben.

2 Grundlagen und verwandte Arbeiten

Prüfungsleistung bieten eine Rückmeldung über den Leistungsstand der Studenten und Übungen dienen als Vorbereitung auf diese. Doch davor gibt es für die Studenten bis auf Probeklausuren kaum eine Rückmeldung über den eigenen Fortschritt und aktuellen Wissensstand. Um das zu ändern, können sie Lerngruppen bilden und sich gegenseitig helfen oder Fragen in einem Universitätsforum, wie dem Auditorium , stellen oder diese direkt an Mitarbeiter des jeweiligen Lehrstuhls mailen.

Eine weitere Möglichkeit zur fortschreitenden Leistungsüberprüfung ist, sich die Studenten gezielt via Peer Assessment gegenseitig bewerten und helfen zu lassen. Eine Herausforderung stellt dabei das Match-Making, die Zuordnung von Studenten zu Paaren oder Gruppen, dar. Im folgenden werden zunächst in Abschnitt 2.1 Anforderungen an ein solches Match-Making diskutiert. Im Anschluss dazu sollen vorhandene Protokolle und Algorithmen vorgestellt und auf ihre Anwendbarkeit für das Match-Making zwischen Studenten hin untersucht werden. In Abschnitt 2.5 wird schließlich eine Möglichkeit zur Erstellung einer Rangliste für Studenten vorgestellt, die das Match-Making unterstützen soll.

2.1 Anforderungen an Match-Making zwischen Studenten

Lerngruppen bieten eine Möglichkeit der gegenseitigen Hilfe und Wissensverfestigung, indem Studenten, die den Lehrstoff verstanden haben, es den Anderen erklären. Doch solche Lerngruppen bilden sich oft erst am Ende eines Semesters zur Prüfungsvorbereitung und sie bestehen zumeist aus Personen, die ein freundschaftliches Verhältnis zueinander haben. Eine solche Gruppe kann daher auch aus Studenten bestehen, die dieselben Probleme mit dem Lehrstoff haben und sich somit nicht gegenseitig helfen können. Eine weitere ungünstige Konstellation wäre ein mittelmäßiger Student, der einem oder mehreren Kommilitonen hilft und dadurch selbst in einen Lernverzug kommt. Finden sich Studenten mit hohem Wissensstand zusammen, können sie ihr Wissen nicht dadurch verfestigen, es anderen erklären zu müssen.

Das Match-Making zwischen Studenten soll diese Probleme umgehen. Leistungsmäßig Schwache, sollen nicht anderen leistungsmäßig schwächeren Studenten helfen müssen. Ebenso sollen mittelmäßige Studenten nicht zu sehr belastet werden, wenn sie einen leistungsmäßig schwächeren Studenten zugeordnet werden. Leistungsstarke Studenten sollen den Schwachen helfen und dadurch die Rolle eines Lehrenden übernehmen, was bei ihnen zur Wissensverfestigung führt. Um so eine Zuordnung vornehmen zu können, bedarf es

einer Rangliste, in der die Studenten nach ihrem Leistungsstand geordnet sind. Beim Match-Making sollen anhand dieser Rangliste der erste, beste, Student dem letzten, schlechtesten, Studenten zugeordnet werden. Der zweite Student der Rangliste soll mit dem Vorletzten ein Match ergeben und so weiter. Dies soll eine gerechte Zuordnung gewährleisten. Die Punkte für die Ranglist können durch unterschiedliche Aufgaben innerhalb der Lehrveranstaltung gewonnen werden und zu Beginn dieser hat jeder Student den gleichen Wert.

Bei Aufgaben die einen gewissen Zeitraum zur Bearbeitung in Anspruch nehmen, gibt es Studenten, die schon eher fertig sind und solche, die Mühe haben, die Zeitvorgabe einzuhalten. Hier sollen ein Student, sobald er fertig ist, einem Studenten zugeordnet werden, der mit der Aufgabe Probleme hat und bisher am wenigsten geschafft hat. Es wäre möglich, sich hierbei an der Rangliste zu orientieren oder aber Meilensteine einzuführen, die abgearbeitet werden müssen. Das Match-Making würde dann einen Studenten, der den letzten Meilenstein geschafft hat, einem zweiten Studenten zuordnen, der die wenigsten der Meilensteine geschafft hat.

Durch diese gegenseitige Hilfe sollen Studenten ihren Wissensstand überprüfen und vergrößern können, wenn sie zu den Leistungsschwachen gehören, beziehungsweise verfestigen können, wenn sie zu den Leistungsstarken gehören und die Rolle des Lehrenden übernehmen. Somit soll gleichzeitig auch die Motivation der Studenten gesteigert werden. Zusätzlich können Hilfsanfragen gestellt oder Hilfe angeboten werden. Die Zuordnung von Anfrage zu Hilfe kann auch über die Rangliste vorgenommen werden. Der schlechteste, Hilfe suchende Student wird damit dem besten, Hilfe bietenden Studenten zugeordnet.

Zusammenfassend:

- Das Match-Making soll den besten Student dem schlechtestem Studenten zuordnen. Ausgehend von einer Rangliste der erste Student hat ein Match mit dem letzten, der zweite mit dem vorletzten usw..
- Beim Matching für praktische Aufgaben soll, sobald ein Student mit der Aufgabe fertig ist – muss nicht Platz 1 auf der Rangliste sein – , dieser dem Studenten zugeordnet, der am langsamsten mit der Aufgabe voran kommt.
- Die Punkte für die Rangliste sollen im Rahmen der Vorlesung oder des Seminars erarbeitet werden können.
- Meilensteine können optional eingeführt werden.
- Hilfsanfragen und Hilfsangebote von und für Studenten sollen gestellt werden können.
- Das Wissen und die Motivation der Studenten soll gesteigert werden.

2.2 Peer-to-Peer und DHT

Ein weit verbreitetes und oft genutztes Protokoll zum Teilen und Verbreiten von Daten, sowie Informationen, welches Match-Making zwischen den einzelnen Teilnehmern praktiziert, ist das Peer-to-Peer-Protokoll mit Nutzung von Distributed Hash Tables (DHT). Im folgenden wird dieses Protokoll kurz vorgestellt und auf seine Tauglichkeit für das Match-Making zwischen Studenten untersucht.

In "Computernetzwerke" von A. Tanenbaum und D. Wetherall [TW12] werden Peer-to-Peer-Netze als Zusammenschluss von vielen einzelnen Rechnern beschrieben, die keine Server in Rechenzentren sein müssen, denn einfache Heimcomputer haben genügend Leistung um sich als Peer zu beteiligen. Die Bezeichnung Peer wird deshalb genutzt, da jeder Teilnehmer sowohl Client als auch Server sein kann. Das heißt, jeder Peer kann sowohl Daten von anderen Peers herunterladen, wie auch selbst Daten zum Download zur Verfügung stellen. Um effektiver zu werden, können zentrale Strukturen vermieden werden. So können z. B. über Distributed Hash Tables (DHT) die Peers direkt miteinander in Verbindung treten.

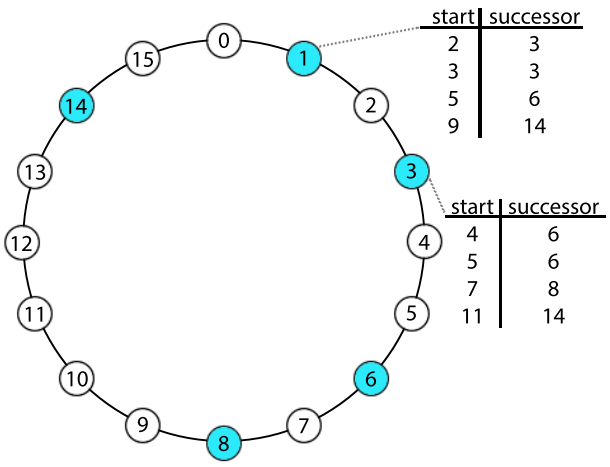


Abbildung 2.1: Ein DHT-Chord-Ring mit realen Knoten (blau), Abbildung nach [TW12]

In solchen strukturierten Peer-to-Peer-Netzen können Gruppen von Peers, Schwärme genannt, die bestimmte Inhalte teilen, mit Hilfe von DHT effizient gefunden werden. Abbildung 2.1 zeigt eine verbreitete Art von DHT: ein Chord-Ring. So ein Ring besteht aus n Knoten, also n IP-Adressen, hier sind es 16. Jeder Knoten hat einen Identifikator, einen m-Bit Hash-Wert, aber nicht jeder Knoten muss für einen real vorhandenen und erreichbaren Computer stehen. In der Abbildung sind diese realen Knoten blau. Jeder dieser Knoten kennt seinen Vorgänger und seinen Nachfolger, beide müssen real sein. Mit Hilfe der Funktion *successor(k)* kann der reale Nachfolgeknoten zu jedem Hashwert k gesucht werden. So können relativ einfach Knoten in den Ring eingefügt werden oder denselben verlassen.

Für alle Inhalte, die zur Verfügung gestellt werden, werden Schlüssel-Wert-Paare in den Knoten gespeichert. Der Schlüssel berechnet sich dabei aus dem Hashwert eines Inhaltsnamens, während der Wert die IP-Adresse ist, die den Inhalt anbietet. Der Knoten der dieses Paar speichert ist immer der reale Nachfolger des Wertes des Schlüssels (*successor(Schlüssel)*). Die Suche erfolgt ähnlich: zu einem Inhaltsnamen wird der Schlüssel erzeugt und der Nachfolgeknoten gesucht. Dieser liefert als Ergebnis eine Liste von Peers. Beginn der Suche ist immer der erste Knoten des Netzes. Um das Suchen effizient zu gestalten brauchen die Knoten eine gute Datenstruktur. Dazu werden Fingertabellen mit mehreren Einträgen genutzt. Zu jedem Startwert wird der reale nachfolgende Knoten gespeichert. In der Abbildung 2.1 sind die Fingertabellen für die Knoten 1 und 3 zu sehen.

Verteilte Systeme, wie Peer-to-Peer-Netze, eignen sich gut, um Inhalte zu verbreiten oder zu suchen. Jedoch wird in einem Netz aus Studenten, für die der Lehrstoff höchstwahrscheinlich selbst neu ist, wohl kaum jemand Inhalte oder Hilfe anbieten. Im Gegensatz dazu wird die Suche danach die wenigen Angebote weit übersteigen. Das bietet weder dem Anbieter noch dem Sucher ein gutes Erlebnis, so dass zunächst einige und später wohl immer mehr Studenten wohl nicht mehr daran partizipieren würden. Des Weiteren muss eine externe Struktur zur Speicherung der Leistung der Studenten geschaffen werden, um sicher zu gehen, dass den schlechtesten Studenten zu erst geholfen wird. Dynamisches Matching, bei dem ein Student,der schon fertig ist, dem schlechtesten noch arbeiteten Studenten zugeordnet wird, ist damit sehr kompliziert. Alles in allem ist das Peer-to-Peer-Protokoll für das Matching zwischen Studenten ungeeignet.

2.3 vektorbasiertes Match-Making

Im Bereich des E-Learnings wird das Match-Making oft mit Hilfe von Vektoren gelöst, z. B. in [YSH03] und [Yan+06]. In diesen Vektoren werden die Interessen der einzelnen Peers (Lernenden) gespeichert. Die Zuordnung zu Lerngruppen oder Partnern erfolgt anschließend nach Ähnlichkeit der Interessen der Peers. Anders kann beim Matchmaking durch Nutzung von Matritzen die Zuordnung beliebig definiert werden: z. B. nach einem Fertigkeitenscore wie in [Cre+05]. Diese beiden Möglichkeiten werden im Folgenden kurz vorgestellt.

In ”A novel self-organizing E-Learner community model with award and exchange mechanisms” [YSH03] wird das Fehlen von Interaktion zwischen den einzelnen Lernenden und die Passivität derselbigen durch den Frontalunterricht im Bereich des E-Learning bemängelt und ein Ansatz vorgestellt, Interaktion durch Gruppierung der Lernenden nach Leistung und Vorlieben herzustellen. Dabei wird der Fokus auf die Zuordnung der Lernenden zu Gruppen mit ähnlichen Interessen und Vorlieben, aber gegensätzlichen Fähigkeiten gelegt. Somit sollen die Schüler sich gegenseitig helfen können und wollen. Dafür wird für jeden Lernenden ein Learner Agent (LA) und für jede Gruppe (Community) einen Group Agent (GA) etabliert. LAs sind 3-Tupel bestehend aus einer Menge von Fähigkeiten C, einer Fähigkeiten-Auswertungs-Tabelle (CET) und einer Interessentabelle (PT). Beide Tabellen

bestehen aus den Spalten Typ – damit ist der Lehrstoff gemeint – und Wissenstand. So wird jedem Lernenden für jeden Lehrstoffbereich sein jeweiliger Wissensstand zugeordnet. Die CET hat zusätzlich eine Spalte für die Auswertung bzw. Note. Die Interessentabelle hingegen hat eine weitere Spalte für den Wert des Interesses. Somit kann für jeden Typ die Fähigkeit und das Interesse des Schülers gespeichert werden. Ein GA ist ebenso ein Vektor, der aus einer Typmenge T – Menge des Lehrstoffes – , einer Hierarchie der Fähigkeiten und einer Tabelle mit LAs, die ihm initial oder über Zeit hinweg zugeordnet wurden.

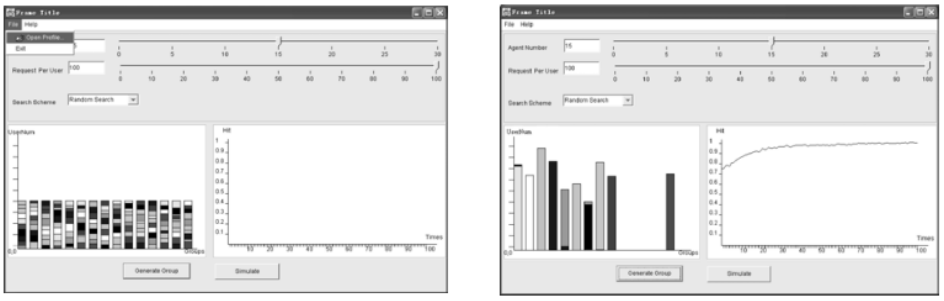


Abbildung 2.2: Initiale Verteilung von LAs auf GAs (links) und die Verteilung nach 100 Anfragen pro LA, aus [YSH03]

Ein GA vermittelt Anfragen zwischen den LAs in seiner Verwaltung und interagiert mit den anderen GAs, um Anfragen von fremden LAs anzunehmen oder Anfragen von eigenen LAs weiterzuleiten, falls sie nicht innerhalb der Gruppe beantwortet werden können. Der LA such mit den Anfragen passende Materialien für den Schüler und zeichnet seine Fähigkeiten auf. Dadurch wachsen die Tabellen. Die Gruppen werden zunächst zufällig festgelegt, wie in Abbildung 2.2 gezeigt, also LAs zufällig auf GAs verteilt. Bei Anfragen, die Interaktion der GAs untereinander erfordern, können LAs dem antwortenden GAs übergeben werden. Somit tauschen die GAs nach der Initialisierung immer wieder LAs aus und ordnen sich über die Zeit selbst. Dieses Etablieren der Gruppen dauert, aber danach bleiben die Gruppenstrukturen fest.

Diese Variante des Match-Making ermöglich zwar das Matching von Studenten mit unterschiedlichem Leistungsstand, jedoch dauert es lange bis sich Gruppen gebildet haben. Da Lehrveranstaltungen zumeist auf ein Semester beschränkt sind, bleibt nicht genug Zeit solche Gruppen etablieren zu lassen. Des Weiteren kann der Lehrstoff einer solchen Veranstaltung kaum feiner unterteilt werden und somit sind die Vektoren und Tabellen zu komplex für die in dieser Arbeit betrachteten Anwendungsfälle. Die Betrachtung ähnlicher Interessen und die Gruppierung nach jenen ist für den Fortschritt im Studium, namentlich das Bestehen von Prüfungen, eher hinderlich. Die Gruppierung von Studenten, welche am Lehrstoff nicht interessiert sind, kann problematisch werden. Insgesamt ist dieses Konzept für das Match-Making zwischen Studenten nicht geeignet.

Die Variante des Matching mit Hilfe von Matritzen, beschrieben in ”An Algorithm for Peer Review Matching Using Student Profiles Based on Fuzzy Classification and Genetic

Algorithms" [Cre+05], soll hingegen einfach konfigurierbar und intuitiv sein. Der Anwendungsbereich dafür ist die sich gegenseitige Überprüfung der Studenten. Die Korrektheit der Zuordnungen für so ein Review sollen durch Eindeutigkeit, das heißt k verschiedene Reviewer, Load Balancing, also die etwa ähnliche Belastung für alle Studenten, den Selbstausschluss und weitere mögliche Zusatzbedingungen gewährleistet werden.

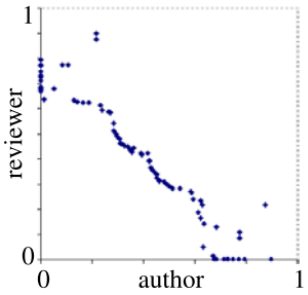


Abbildung 2.3: Matching von Studenten zu Arbeiten nach Leistungsstand, aus [Cre+05]

Das Matching von einem oder mehreren Reviewern zu einem Autor wird unter pädagogischen Aspekten optimiert, um das gegenseitige Helfen und Erklären in den Vordergrund zu rücken. Dieses kann durch komplementäres Matching nach einem Fertigkeitenscore oder nach Lerntyp vorgenommen werden. Beim komplementären Matching wird den Studenten ein Wert zwischen 0 und 10 zugeordnet, wobei 0 die schlechteste und 10 die beste Fertigkeit symbolisiert. Ein Match erfolgt bei einem möglichst gegensätzlichen Fertigkeitenscore, siehe Abbildung 2.3. Bei den Lerntypen werden die Studenten zwischen sequentiell und global verortet und ähnliche Lerntypen werden gematcht. Für beide Fälle werden Prototypen definiert, die jeweils optimale und schlechteste Matches angeben. Die Spanne der Studenten liegt dazwischen und die Zuordnung erfolgt so, dass die Matches eher an den optimalen als an den schlechtesten Prototypen liegt. Das Matching wird durch Unschärfe (Fuzziness) gewichtet und durch einen genetischen Algorithmus weiter optimiert. Dabei werden für Maps, die Matritzen für die Zuordnung von Reviewern zu Autoren, ebenso die Nähe zu optimalen Prototypen berechnet. Mit Hilfe von Evolution durch den Austausch zweier Teile (Enden) verschiedener Maps an einem zufälligen Breaking-Point werden neue Möglichkeiten für das Matching kreiert, die wiederum an der Nähe zu den optimalen Prototypen bemessen werden.

Auch dieses Match-Making erfüllt die Anforderung des Matchings von Studenten mit unterschiedlichem Leistungsstand. Jedoch erfolgt die Zuordnung ausschließlich von Studenten zu eingereichten Arbeiten, welches lediglich einen Anwendungsfall darstellt und ist nicht auf Paarung oder Gruppierung von Studenten anwendbar. Des Weiteren kann es, wie in Abbildung 2.3 auf Grund der Verwendung von der Nähe zu Prototypen und Unschärfe dazu führen, dass der Student mit dem größten Leistungsdefizit nicht dem besten zugeordnet wird, sondern einem mittelmäßig guten, der wiederum mit der Aufgabe der Hilfe somit leichter überfordert sein kann. Auch interaktives Matching bei praktischen Aufgaben ist nicht möglich. Das heißt Studenten, die z. B. eine Programmieraufgabe gerade beendet

haben, können nicht sofort einem Studenten zugeordnet werden, der noch nicht fertig ist. Insgesamt ist diese Art des Match-Makings für die Breite der Ansprüche an ein Matching zwischen Studenten nicht ausreichend.

2.4 Match-Making in Computerspielen

Ein weiterer Bereich, in dem Match-Making benötigt wird, sind Computerspiele. Während bei Singleplayer-Spielen der Schwierigkeitsgrad oftmals manuell vom Spieler angepasst werden kann, sollten bei Multiplayer-Spielen mit Player-versus-Player-Elementen (PvP) Spieler mit einem ähnlichem Fertigkeitenlevel zusammenspielen. Dadurch bleibt das Spielen eine Herausforderung und wird nicht langweilig, weil ein Spieler oder eine Gruppe vom Gegner dominiert wird oder umgekehrt. In League of Legends (LoL) wird die Fertigkeit eines Spielers durch eine Zahl, das Match-Making-Rating (MMR), dargestellt, welche durch alle vorherigen Spielergebnisse bestimmt wird [Gam16]. Im folgenden wird das Match-Making in diesem Spiel genauer betrachtet und weitere dafür genutzte Variablen vorgestellt.

In ”Matchmaking in multi-player on-line games: studying user traces to improve the user experience” [VMM14] wird das Match-Making im Mulitplayer Online Battle Arena (MOBA) Spiel League of Legends auf Grund von mehr als 28 Millionen LoL-Sessions analysiert. Dabei wird die Wichtigkeit eines guten Matchings von Spielern in Mutiplayer Online Games herausgestellt und dass es trotzdem öfters zu einem unausgewogenen Matching kommt. Das Match-Making in LoL berücksichtigt mehrere Faktoren. Ein Faktor ist die Zeit zwischen dem Beginn der Suche nach Mitspielern und dem eigentlichen Spielbeginn soll kurz sein, so dass sich Spieler nicht zu lange in der Warteschlange befinden. Die durchschnittliche Wartezeit beträgt 90 Sekunden und steigt bei sehr guten Spielern. Um das Spielerlebnis zu verbessern wird versucht die durchschnittliche Wartezeit auf 30 Sekunden zu verkürzen. Ein weiterer zu berücksichtigender Faktor ist die Latenz von Client zu Spielserver, da es in LoL auf schnelle Reaktionen der Spieler ankommt und Mitspieler mit zu hoher Latenz einen negativen Effekt auf das Spielerlebnis haben können. Das kann das durch Statistiken bestimmte Matching verfälschen. Dieses Match-Making ergibt sich aus der Anzahl schon gespielter Spiele je Spieler, der Größe des Premades (eines vorab zusammengestellten Teams) und der Spieler-Ranking. Ein Premade wird als Vorteil gesehen und je nach der Größe dieser Gruppe, werden entweder Premades als Gegnerteam gewählt, die über eine ähnliches Fähigkeiten-Level als Gruppe verfügen oder es werden Spieler ins gegnerische Team gematcht, die, einzeln betrachtet, das Fähigkeiten-Level des Premades überschreiten.

Das Ranking ist das zentrale Element in LoL. Dabei werden die Spieler in Ligen und Divisionen unterteilt, um ein überschaubares Umfeld zu haben und Aufstiegs- und somit Spielanreize zu geben. Für das Ranking wird eine modifizierte Version des Elo Rating System genutzt, welches ursprünglich zur Bestimmung des relativen Levels eines Schachspielers entwickelt wurde.

$$Ea = \frac{1}{1 + 10^{\frac{Rb-Ra}{400}}}$$

Die Wahrscheinlichkeit, dass Spieler A ein Spiel gewinnt (Ea), ist für je 400 Punkte Unterschied im Ranking (R) zu Spieler B um ein zehnfaches höher oder niedriger, je nach dem, wer das höhere Ranking hat. Diese Berechnung ist für Schach üblich und kann in LoL modifiziert worden sein [VMM14].

$$Ra_{\text{new}} = Ra_{\text{old}} + K \cdot (Sa - Ea)$$

Nach jedem Spiel ändert sich das Ranking wie folgt: der neue Wert setzt sich aus dem alten Ranking-Wert (MMR) und der Punktzahl für das gerade abgeschlossene Spiel zusammen. Sa ist dabei der Ausgang des Spiels für Spieler A, 1 für Gewinn und 0 für eine Niederlage. Davon wird die vorher berechnete Wahrscheinlichkeit für einen Sieg Ea abgezogen und mit einem persönlichen Adjustierungsfaktor K multipliziert. So bekommt Spieler A für ein Spiel, bei dem die Wahrscheinlichkeit zu gewinnen hoch war, weniger Punkte als für ein Spiel, bei dem das System eine Niederlage für wahrscheinlicher hielt. Der Adjustierungsfaktor K beginnt bei 25 für die ersten circa 30 Spiele, um für den neuen Spieler schnell eine grobe Position im Ranking zu bestimmen, und nimmt danach stetig ab auf 15 bis 7. Zusätzlich bekommen neue LoL-Spieler einen Start-Elo-Wert von 1200 für die ersten 10 Spiele.

Die Analyse der Daten aus [VMM14] zeigt, dass für das Matching in Computerspielen die Fairness der wichtigste Aspekt ist. Dies wird in LoL neben dem Matching bei ähnlichem MMR auch durch die Anzahl absolvierter Spiele versucht. Nebenbei muss bei solchen Computerspielen auch die Latenz berücksichtigt werden. Dennoch könnte das Match-Making nach Meinung der Autoren durch zusätzliche Kriterien eventuell verbessert werden.

Der Ansatz eines Matchings über ein Ranking-System ist auch für ein Match-Making zwischen Studenten interessant. Anders als bei Computerspielen sollten jedoch keine Gruppen mit ähnlichem Ranking gebildet werden, sondern Studenten mit sehr hohem Ranking sollen Studenten mit sehr niedrigem Ranking zugeordnet werden. Dies wäre anhand einer Rangliste sehr einfach zu realisieren. Auch der Erwerb von Punkten für eine solche Rangliste muss anders gestaltet werden. Aber ein solches System bietet die Möglichkeit eine Vielzahl von Faktoren zu berücksichtigen. Die Anzahl der absolvierten Spiele, sowie die Anzahl der Spiele in letzter Zeit sind solche zusätzlichen Faktoren. Auf das Matching von Studenten übertragen könnte Anzahl absolvierter Aufgaben eines Studenten ein solcher zu berücksichtigender Faktor sein. Die Latenz hingegen kann vernachlässigt werden, da sich die Studenten zumeist in der Stadt befinden, in der sich auch die Hochschule befindet und die Aufgaben und das Matching in Vorlesungen oder Seminaren ausgeführt werden. Alles in allem kann das Match-Making in Spielen wie LoL eine Vorlage für das Matching zwischen Studenten bieten.

2.5 Gamification

Wenn das Match-Making in Computerspielen als Grundlage für das Matching von Studenten dient, kann auch über die Nutzung von Gamification nachgedacht werden. Dabei werden Elemente aus Spielen übernommen und in einem Spiel-fremden Kontext verwendet. Ziel ist es, dadurch die Motivation und die Teilnahme der Zielgruppe zu erhöhen. Das Lernen an sich und dadurch insbesondere Lernplattformen sind dafür geeignet, da durch Punkte für erledigte Aufgaben, Noten, etc. schon Elemente von Spielen vorhanden sind. Das Auditorium, eine Frage-und-Antwort-Plattform für Studenten der Fakultät Informatik der TU Dresden, die ähnlich wie StackOverflow funktioniert, nutzt Gamification [Bei14].

Auch für das Match-Making zwischen Studenten kann Gamification genutzt werden. Da eine Wertung mit Punkten schon für das Matching nötig ist, können diese auch dazu verwendet werden um dem Studenten Feedback über seinen Leistungsstand zu geben. Aus diesen Punkten wiederum lassen sich Level berechnen, die einen Fortschritt im Lernprozess anzeigen und somit die Studenten motivieren können. Bei Spielen wird die Hauptaufgabe in viele kleine Unteraufgaben aufgeteilt, die einfacher zu lösen sind und ein Gefühl des merkbaren Vorankommen für den Spieler bieten. Die Hauptaufgabe, die Prüfung am Ende des Semesters zu meistern, kann ebenso durch kleine vorbereitende Aufgaben leichter werden und diese Unteraufgaben bieten den Studenten die Möglichkeit, sich zu testen und zu verbessern. Weitere Elemente in Gamification sind Abzeichen, Badges genannt, und Leaderbords. Die Verwendung letzterer kann jedoch einen negativen Effekt auf die Motivation der Studenten haben und auch Abzeichen können falsche Anreize setzen, siehe [Bei14]. Andere Elemente von Gamification wie zum Beispiel die soziale Interaktion, sind auch schon teilweise in der Struktur des Match-Makings bedingt oder sind zwar für das Gestalten eines Programms wichtig, würden in dieser Arbeit jedoch zu weit führen, da der Fokus auf dem Match-Making zwischen Studenten liegt.

3 Anwendungsfälle

Die Zuordnung von leistungsstarken Studenten zu leistungsschwächeren, um sich gegenseitig zu helfen und das Verständnis für den Lehrstoff zu steigern, ist in vielen Bereichen des universitären Lernens möglich. Dieses Match-Making kann wie in Kapitel 2 beschrieben anhand von Punkten in einer Rangliste vorgenommen werden. Die Punkte müssen dabei von Studenten im Laufe der Lehrveranstaltung erarbeitet werden. Anhand dessen kann gleichzeitig eine Lernkurve über das Semester für die Studenten erstellt werden. In diesem Kapitel werden Anwendungsfälle für das Sammeln von Punkten und das Match-Making zwischen Studenten in verschiedenen Lehrformen vorgestellt.

3.1 während der Vorlesung

Vorlesungen sind ein Hauptbestandteil des Studiums. Hier wird der neue Lehrstoff vermittelt, der in Übungen verfestigt wird. In wie weit die Studenten den Stoff verstanden haben und anwenden können, wurde bisher zumeist erst am Ende eines Semesters in einer Prüfung ermittelt. Nur in wenigen Vorlesungen werden den Studenten praktische Testaufgaben oder ähnliches angeboten. Ein Beispiel dafür wäre der Praktomat vom Lehrstuhl für Softwaretechnologie [Mor]. Gegenseitige Hilfe und Erklären von Sachverhalten zwischen Studenten kann das Verständnis für den Lehrstoff steigern. Um die Studenten einander dafür optimal zuzuordnen, muss zunächst eine Basis für einen Leistungsvergleich geschaffen werden. Dies kann als Punkte in einer Rangliste ausgedrückt werden.

Solche Punkte können durch verschiedene Aufgaben sogar während einer Vorlesung gesammelt werden. Eine mögliche Aufgabe sind Multiple Choice Tests. Das Wissen, welches im Laufe der Vorlesung erworben wurde kann somit direkt und schnell abgefragt werden. Die Studenten bekommen eine Rückmeldung über ihren Leistungsstand und die dadurch erworbenen Punkte verorten sie in der Rangliste. Dabei ist jedoch zu beachten, dass es verschiedene Arten von Multiple Choice Tests gibt und jede Art bestimmte Risiken für Verfälschungen gibt, siehe [Krü13]. Bei Aufgaben, wo nur eine Antwortmöglichkeit richtig ist, müssen die falschen Antwortmöglichkeiten, Distraktoren genannt, ebenso plausibel sein, wie die richtige. Auch müssen genügend Distraktoren vorhanden sein, ansonsten können Studenten zu oft zufällig richtig wählen, obwohl sie die Antwort nicht wissen. Bei Aufgaben, wo mehrere der Antwortmöglichkeiten richtig sind, werden falsch gewählte Antworten negativ gewertet. Das kann sich durchaus auf die Punkte und somit die Rangliste auswirken.

Punkte können auch durch offene Fragen oder kleine Aufgaben, wie kurze Textaufgaben oder kleine Programmieraufgaben, in denen nur nach wenigen Zeilen Code gefragt wird, gesammelt werden. Dabei bietet es sich an, dass sich die Studenten gegenseitig bewerten. Somit wiederholen sie den Stoff, finden mögliche Fehler und können mit wenig Aufwand für Mitarbeiter Punkte sammeln. Die Kontrolle kann dabei durch Vorlesenden erfolgen, der die Lösung der Aufgabe abschließend an der Tafel zeigt oder von einem Studenten zeigen lässt.

Einige Vorlesungen sind Teil einer Vorlesungsreihe und bauen aufeinander auf, wie z. B. Mathe 1, 2 und 3. Im Falle solcher zusammenhängender Vorlesungen könnte der Punktestand aus der Vorlesung des vergangenen Semesters übernommen oder aus der vorhergehenden Prüfung berechnet werden. Da einige Studenten Vorlesungen mehrmals belegen könnte das allerdings bei sich ändernden Aufgaben und somit einer anderen Rangliste zu Problemen kommen. Der Student könnte zu gut oder zu schlecht bewertet werden. Um diesen Faktor so klein wie möglich zu halten könnte eine Konvertierungsvorschrift geschaffen werden. Das soll hier jedoch nicht weiter betrachtet werden.

Für das Matching innerhalb von Vorlesungen gibt es ebenfalls verschiedene Anwendungsmöglichkeiten. Zunächst wie schon für die offenen Fragen und kurzen Aufgaben angedacht und auch für Multiple Choice möglich: ein kontrollierendes Matching. Dabei werden die Studenten einander zugeordnet, nachdem alle mit der Bearbeitung der Aufgabe fertig sind. Der beste Student, der auf der Ranglist die meisten Punkte hat, bekommt hierbei die Antwort vom Studenten mit den wenigsten Punkten. Der zweite Student auf der Rangliste wird mit dem vorletzten gematcht und so weiter. Ziel ist es, dass die Studenten gegenseitig ihre Fehler berichtigen beziehungsweise von richtigen Antworten lernen. Eine zusätzliche Instanz, welche die korrekten Antworten bereit hält, muss dabei verfügbar sein, damit Fehler sich nicht verfestigen.

Eine andere Art des Match-Making ist das sofortige Matching, welches ebenso bei kurzen Aufgaben, wie Programmier- und Rechenaufgaben genutzt werden kann. Dabei wird ein Student sofort, nachdem er eine Aufgabe beendet hat, dem Studenten mit den wenigsten Punkten auf der Rangliste zugeordnet, dem noch kein anderer Student zugeordnet wurde. Dadurch können Studenten, die gut mit einer Aufgabe zurecht kommen, obwohl sie nicht die besten Studenten des Faches sein müssen, schnell ihren Kommilitonen helfen. Dem Studenten mit den wenigsten Punkten wird dabei zuerst Hilfe angeboten bekommen, da es am wahrscheinlichsten ist, dass er sie braucht.

Eine dritte Anwendung für das Match-Making in Vorlesungen ist das Gruppenmatching für Diskussionen, Lerngruppen, etc. Auch kleine Aufgaben können in Gruppen gelöst werden. Dabei ist sowohl die Anzahl der Teilnehmer einer Gruppe, wie auch ihre Zusammensetzung wichtig und muss variabel gestaltbar sein. Um die Belastung der Studenten in den verschiedenen Gruppen möglichst gleich zu halten und den Lerneffekt für schwächere Studenten gleichzeitig zu erhöhen müssten verschiedene Gruppenzusammensetzungen ausprobiert und bewertet werden. Um das Match-Making grundlegend zu testen wird hier folgende Zuordnung vorgeschlagen: in eine Gruppe kommt jeweils ein Student vom oberen

und vom unteren Ende der Rangliste. Aufgefüllt wird diese Gruppe mit Studenten aus der Mitte der Rangliste. Für die verbleibenden Studenten wird ebenso verfahren.

3.2 begleitend zu Übungen

Übungen sind immer ein Teil einer Lehrveranstaltung. Das heißt zu vielen Vorlesungen gibt es regelmäßige Übungseinheiten. Dabei werden Übungsaufgaben von einem Tutor kontrolliert und der Lösungsweg aufgezeigt. Die Studenten sollten die Aufgaben vor der jeweiligen Übungseinheit lösen, was jedoch nicht immer der Fall ist. Übungen können also als eine Erweiterung der Vorlesung betrachtet werden und dementsprechend gelten die erworbenen Punkte in der Vorlesung auch in der zugehörigen Übung.

Auch wenn sich Übungen scheinbar anbieten um weitere Punkte für die Vorlesung zu sammeln und somit schneller eine aussagekräftigere Rangliste zu schaffen, bringt ihre Struktur doch ein paar Probleme mit sich. So ist die Zeit in einer Übungsstunde damit ausgelastet, dass die gestellten Aufgaben besprochen, das heißt kontrolliert, werden. Möglich wäre es Punkte zu sammeln in dem die Aufgaben vorher im zu entwickelnden Programm gelöst werden. Jedoch bereiten sich einige Studenten nicht auf die Übung vor und versuchen sich erst gar nicht daran, die gestellten Aufgaben zu lösen. Würden zu erwerbende Punkte genügend Anreiz bieten, dass sich eine größere Menge an Studenten vorab mit den Aufgaben beschäftigt? Ein weiterer Aspekt ist die Kontrolle. Sofortige Rückmeldung über Fehler in den Aufgaben würden den Studenten helfen, jedoch sind die Übungsaufgaben dazu teilweise zu komplex. Außerdem könnten dadurch die Studentenzahlen in den Übungen abnehmen. Wenn andererseits erst in der Übungseinheit kontrolliert wird, haben die Studenten eventuell keinen Anreiz diese Übung online vorzubereiten. All das sollte in weiterführenden Arbeiten untersucht werden.

Für das Matching gibt es auch hier mehrere Möglichkeiten. Falls die Übungen online gelöst werden, können Studenten, welche die Aufgaben schon bearbeiten haben, zu den leistungsschwächsten Studenten zugeordnet werden und als Ansprechpartner für mögliche Probleme mit den Aufgaben dienen. Das kann jedoch dazu führen, dass gute Studenten mit der Lösung der Übung warten oder darauf verzichten, um nicht helfen zu müssen. Eine zweite Möglichkeit für das Match-Making wären Hilfsangebote. Studenten, die sich sicher im Lehrstoff fühlen könnten solche Angebote stellen und im Gegenzug könnten Studenten, die Hilfe benötigen, Hilfsanfragen stellen. Diese Angebote und Anfragen könnten anschließend einander zugeordnet werden. Eine dritte Möglichkeit ist ein Gruppenmatching. Wenn so ein Matching gewünscht ist, können Gruppen, die ähnlich wie Lerngruppen fungieren, gebildet werden. Die Studenten einer Gruppe sollen gemeinsam die Übung vorbereiten. Das Matching für Studenten zu einer solchen Gruppe kann unterschiedlich gelöst werden. Welche Art gewählt wird, folgt später. Solche Lerngruppen versprechen Erfolg, da mehr Studenten Übungsaufgaben vorbereiten würden, der Lehrstoff damit schon innerhalb des Semesters wiederholt wird und die Studenten eigene Schwächen schneller erkennen können.

Alles in allem ist es schwierig in Übungen Punkte zu sammeln. Da Übungen jedoch zu Vorlesungen gehören und in diesen schon Punkte gesammelt werden können, wird dieser Aspekt in dieser Arbeit nicht weiter betrachtet. Das Matching in Übungen ist ebenso problematisch. Am erfolgversprechendsten scheint das Gruppenmatching zu sein. Sich mit den Auswirkungen der anderen Matchingarten zu beschäftigen würde in dieser Arbeit jedoch zu weit führen.

3.3 Programmierpraktika

Während des Informatikstudiums gibt es viele Praktika verschiedener Art, in denen besonders die Programmierkenntnisse geschult werden. Einige davon sind zu komplex für ein Match-Making wie in dieser Arbeit vorgestellt, weil die Aufgaben an sich zu umfangreich für einen Leistungsvergleich zwischen Studenten sind oder weil sich die Aufgaben für die einzelnen Studenten von einander unterscheiden, so dass sie sich gegenseitig nicht mehr ohne viel Mehraufwand helfen können. Haben die Studenten jedoch dieselbe nicht zu komplexe Aufgabe, so kann auch hier ein Matching vorgenommen werden, um den Studenten, die Probleme haben schnell zu helfen.

Punkte zur Leistungsbewertung können je nach Art des Praktikums unterschiedlich gesammelt werden. Bei wöchentlich kontrollierten und voneinander relativ unabhängigen Aufgaben, können die Punkte für die Ergebnisse jede Woche gutgeschrieben werden. Somit wird über das Semester ebenso wie bei den Vorlesungen eine aussagekräftige Leistungskurve über die Teilnehmer aufgebaut. Falls das Praktikum ein Teil einer Vorlesung ist oder auf einer solchen aufbaut, kann auch der Punktestand aus der Vorlesung übernommen werden oder sich aus der Prüfungsleistung ergeben. Für fortlaufende Praktika in denen ein zusammenhängendes Programm geschaffen werden soll, bietet sich eine dritte Möglichkeit an. Der Fortschritt kann durch Meilensteine in Form von Abzeichen, auch Badges oder Achievments genannt, dargestellt werden. Das Matching würde sich in diesem Fall nicht an der Punktzahl ausrichten sondern an der Geschwindigkeit des Fortschritts der Studenten.

Ein solches Match-Making auf Grund von Meilensteinen ordnet einen Studenten, sobald dieser einen bestimmten Grad des Fortschritts, also einen spezifischen Meilenstein, erreicht hat, dem Studenten zu, der am langsamsten vorankommt, also bisher die wenigsten Meilensteine erreicht hat. Bei Praktika in denen Punkte vergeben werden, kann der schnellste Student dem leistungsschwächsten zugeordnet werden, wie schon in 3.1 dargestellt. Des Weiteren wäre ebenso ein Gruppenmatching möglich, dass auf der Grundlage von Punkten aus einer zugehörigen Vorlesung vor dem Start des Praktikums berechnet werden kann. Bei Praktika, für die es jede Woche voneinander unabhängige Aufgaben gibt, können solche Gruppen auch noch im Laufe des Semester als Lerngruppen gebildet werden.

4 Konzeption und Befragung

Ein Programm zum Matching von Studenten soll bei selbigen Zuspruch und dauerhafte Verwendung finden. Daher erfolgt als Grundlage für die Konzeption eines solchen Programms eine kurze Befragung der Studenten, welche den Zuspruch messen und zusätzliche Wünsche an ein solchen Programm erfassen soll. Aufbauend auf dieser Befragung, der Vorbetrachtung in Kapitel 2 und den in 3 dargestellten Anwendungsfällen, wird anschließend eine Anforderungsanalyse durchgeführt. Auf deren Grundlage wird eine Konzeption mit dem Schwerpunkt auf dem Matching erstellt. Abschließend wird eine Befragung zu der Konzeption vorgenommen, um eventuelle Fehler darin zu erkennen und vorgeschlagene Verbesserungen aufzunehmen.

4.1 Kurzbefragung

Eine Möglichkeit grob den Zuspruch zu einem Programm zu messen, ist die Nutzung der System Usability Scale (SUS) nach [Bro86]. Der Fragebogen der SUS enthält 10 Fragen mit einer 5-wertigen Likert-Skala von 1 für "stimme gar nicht zu" bis 5 für "stimme voll und ganz zu" als Antwortmöglichkeit und misst die wahrgenommene Benutzerfreundlichkeit. Dafür werden den Antworten Zahlenwerte zugewiesen und diese anschließend aufsummiert. Somit ergibt sich eine Skala von 0 bis 100, wobei 100 die perfekte Usability darstellt. Ein häufiges Problem in solchen Umfragen ist die Tendenz von Probanden generell zustimmend oder ablehnend zu antworten. Um solche Verzerrungen ausschließen zu können, werden einige Fragen invertiert, das heißt negativ formuliert. An diesem Fragebogen soll sich die Kurzbefragung orientieren.

Die Fragen werden den Probanden häufig erst gestellt, nachdem sie das Programm testen durften. Da in diesen Fall jedoch noch kein Programm existiert, das getestet werden kann, aber die mögliche Nutzung schon abgeschätzt werden soll, wurde der Test vorgezogen und durch eine kurze Beschreibung dessen ersetzt, was das Programm können soll. Des Weiteren wurden die Fragen zum Teil abgewandelt oder durch andere, an das Grundkonzept angepasste Fragen, ersetzt. Zusätzlich wurde eine weitere Frage eingefügt, welche die Wünsche der Studenten an ein solches Programm aufzeigen soll. Dabei sind mehrere Antwortmöglichkeiten wählbar und ergänzbar. Der Fragebogen zu dem Programm sieht insgesamt wie folgt aus:

Das zu entwickelnde Programm, soll in Vorlesungen, Seminaren und Praktika Verwendung finden. Ziel ist, den Lernfortschritt durch Punkte (eventuell Level) und Abzeichen (Badges)

sichtbar zu machen. Punkte sollen durch verschiedene kleinere Aufgaben (z. B. Multiple Choice Test, kurze Programmieraufgaben, ...) erworben werden können. Abzeichen sollen durch Fortschritte bei komplexeren Aufgaben, sowie Interaktion mit anderen erworben werden können. Hat ein Student Probleme mit einer Aufgabe, kann ihm ein anderer, der die Aufgabe schon bewältigt hat, helfen. Diese Zuordnung erfolgt durch Matching.

1. Ich kann mir sehr gut vorstellen, das Programm regelmäßig zu nutzen.

2. Ich finde das Programm-Konzept als unnötig komplex.

3. Ich empfinde das Programm-Konzept als nützlich.

4. Es macht mir nichts aus bewertet zu werden.

5. Ich möchte nicht gematcht werden.

6. Ich finde es gut, dass verschiedene Funktionen (Punkte, Abzeichen, Matching,...) in einem Programm integriert sind.

7. Ich habe mir so ein Programm schon gewünscht.

8. Ich würde eher offline/handschriftlich arbeiten wollen.

9. Ich finde es gut, meinen Lernfortschritt beobachten zu können.

10. Mir reichen die bisherigen Angebote (Übungen, Auditorium) aus.

Zusatz: Ich wünsche mir in einem solchen Programm:
"Punkte"
"Level"
"Abzeichen"
"Lerngruppen-Matching"
"Anderes, und zwar:_____"

Dieser Fragebogen wurde an Studenten in der Vorlesung "Datenbanken (Grundlagen)" zum Ende des Sommersemesters 2016 verteilt. Diese Vorlesung richtet sich an Studenten der Informatik, Medieninformatik, Wirtschaftsinformatik und Informationssystemtechnik und wird üblicherweise im 4. bzw. 6. Semester besucht. Das hat den Vorteil, dass die Studenten fachlich dazu in der Lage seien sollten, auf Grund des kurzen Einleitungstext, sich das zu entwickelnde Programm vorzustellen und einschätzen zu können. Weiterhin haben die Studenten schon Erfahrungen mit anspruchsvollen Vorlesungen und Prüfungen gemacht, sind aber immer noch im Grundstudium und können somit genaue Angaben davon machen, was ihnen helfen würde, zu lernen und sich besser auf Prüfungen vorzubereiten. Insgesamt wurden 78 Fragebögen ausgeteilt und ausgefüllt.

Anhand der Ergebnisse der Fragebögen wurde, ebenso wie bei der SUS, ein Summen-Score ermittelt. Dazu wurden zunächst alle invertiert formulierten Fragen – die 2, 5, 8 und 10 – umkodiert. Das heißt die Fragen wurden umformuliert und die Antworten auf der Likert-Skala invertiert. Somit lassen sich je Fragebogen alle gegebenen Antworten mit ihrem Wert auf der Likert-Skala aufsummieren und damit kann ein Prozentwert an

Zustimmung ermittelt werden. Der Median des Summen-Scores zeigt einen allgemeinen Wert an Zustimmung oder Ablehnung, der bei SUS sowie in dieser Kurzbefragung aber nur als grober Richtwert angenommen wird. Für genauere Analysen sind beide Befragungen nicht ausgelegt. Zusätzlich zur Likert-Skala von Ablehnung bis Zustimmung wurde den Teilnehmern in dieser Kurzbefragung die Möglichkeit gegeben, auch den Wert *"Kann ich nicht beurteilen"* zu wählen. Somit müssen sich Probanden nicht für einen Wert auf der Skala entscheiden, wenn sie die Auswirkungen der Frage auf sie nicht beurteilen können. Für den Summen-Score müssen diese Antworten jedoch entweder mit einem Wert belegt werden, zum Beispiel mit neutralen 3 Punkten – der Mittelwert einer 5-wertigen Likert-Skala – nivelliert, oder die Fragebögen mit solchen Antworten müssen von der Bewertung ausgeschlossen werden. Der Vorteil dabei besteht darin, dass die Probanden selbst schon gezeigt haben, dass sie eine Bewertung nicht abgeben können und somit ihre Antworten oder die Nivellierung dieser das Ergebnis nicht fälschlich beeinflussen könnten.

Im ersten Fall, mit der Nivellierung auf den Mittelwert, konnten 74 von 78 beantworteten Fragebögen in die Berechnung einbezogen werden. Die restlichen 4 mussten aus Gründen, wie mehrere oder gar keine Kreuze bei einer Frage, ausgeschlossen werden. Damit wurde ein Summen-Score von 34 – in einem Wertebereich von 10 bis 50 – erreicht. Das entspricht 60% des Meximalwerts. Werden die Fragebögen, bei denen eine Nivellierung nötig ist, auch ausgeschlossen, ergeben die verbleibenden 49 Fragebögen einen Summen-Score von 36, entsprechend 65%. Das zeigt, dass die Nivellierung der Antworten auf den exakten Mittelwert, den realen Zustimmungswert unter den Studenten in Richtung Mitte, das heißt 50%, verzerren könnte. Der Wert ohne Nivellierungen scheint daher eher repräsentativ zu sein. Zumal das arithmetische Mittel der einzelnen Fragen bis auf eine Ausnahme zwischen 3 und 4 auf der Likert-Skala liegt. Aus diesen Ergebnissen könnte auf eine tendenzielle Zustimmung zu einem solchen Programm und Match-Making unter den Studenten geschlossen werden. Für die SUS wird ab einem Wert von 68% von guter oder überdurchschnittlicher Usability gesprochen. Doch im Gegensatz dazu konnte in diesem Fall kein Programm getestet werden und die Fragen wurden verändert. Somit kann auch der hier erreichte Wert von 65% als Zustimmung und positive Rückmeldung zu dem zu konzipierenden Programm gewertet werden.

Dieses zeigt sich auch in den beiden Extremwerten der Mittelwerte zu den einzelnen Fragen der Kurzbefragung. Frage 7 erhielt den niedrigsten Wert, Frage 9 den höchsten. In der folgenden Tabelle sind die Mittelwerte gezeigt, sie liegen auf einer Skala von 1 (ablehnend) bis 5 (zustimmend):

Frage	$\bar{x}$
7. Ich habe mir so ein Programm schon gewünscht.	2,20
9. Ich finde es gut, meinen Lernfortschritt beobachten zu können.	3,84

Dass nur wenige Studenten sich schon ein solches Programm gewünscht haben (Frage 7), wirkt sich auch negativ auf den Summen-Score aus. Obwohl dies auch dadurch bedingt

sein kann, dass ein solches Programm eben noch nicht – auch nicht in ähnlicher Weise – existiert und die Studenten erst durch die Kurzbeschreibung einen Eindruck von etwas bekommen haben, worüber sie sich womöglich noch keine Gedanken gemacht haben. Somit könnten viele Studenten in der Tat noch nie über ein solches Programm nachgedacht haben und dementsprechend könnten viele es sich nicht schon gewünscht haben. Dafür spricht auch, dass Frage 9 einen hohen Zuspruch bekommen hat. Viele Studenten möchten offenbar ihren Lernfortschritt beobachten können. Dies kann auf unterschiedlichen Wegen realisiert werden, somit kann es sein, dass nur wenige dabei an ein solches Programm, wie das hier zu konzipierende, gedacht haben. Des Weiteren lässt sich aus der Zustimmung zu Frage 9 auch ableiten, dass ein Programm, in dem es möglich ist, seinen Lernfortschritt zu beobachten, gut angenommen werden könnte.

Antwortmöglichkeiten	N	Prozent der Fälle
Punkte	48	72,7%
Level	37	56,1%
Abzeichen	19	28,8%
Lerngruppen-Matching	24	36,4%
Anderes, und zwar...	8	12,1%

Abbildung 4.1: Antworten auf die Zusatzfrage in absoluten Zahlen N und prozentualem Anteil an Antworten.

Mit der Zusatzfrage *"Ich wünsche mir in einem solchen Programm"* konnten die Studenten ihre Anforderungen an das Programm zum Ausdruck bringen. Es konnten mehrere Antwortmöglichkeiten gewählt werden und in einem freien Feld weitere Ideen und Wünsche notiert werden. Da sich diese Frage zusammen mit einigen Demografiefragen auf der Rückseite des Fragebogens befand, wurde sie – obwohl vorher darauf aufmerksam gemacht wurde – von einigen Studenten übersehen. Für die Berechnungen zu dieser Frage wurden daher nur die Fragebögen mit beantworteter Rückseite, insgesamt 66, einbezogen. In Abbildung 4.1 sind die gegebenen Antworten zu sehen, sowie in der letzten Spalte wie viele der 66 Probanden sich die jeweilige Option gewünscht haben. So wünschen sich 72,7% der Studenten, dass erreichte Punkte gezeigt werden und 56,1% möchten ihren Lernfortschritt durch Level angezeigt bekommen. Diese beiden Features sollten daher in das Programm integriert werden. Weniger Zustimmung gibt es für das Lerngruppen-Matching (36,4%). Dennoch sollte es integriert werden, da das Matching allein schon Teil des Programms sein wird. Des Weiteren könnte das Angebot des Lerngruppenmatching auf Freiwillige beschränkt werden. Da Abzeichen nur von 28,8% der Studenten gewünscht wurden, werden sie zunächst nicht berücksichtigt. Eventuell können sie später ergänzt werden, sollte sich dann eine Mehrheit der Studenten dafür aussprechen.

In dem offenen Antwortteil der Zusatzfrage, sowie in dem Bereich für allgemeine Anmerkungen zum Programm haben einige Studenten weitere Wünsche notiert oder Bedenken geäußert. Zwei wünschen sich, dass es Aufgaben ähnlich wie beim Praktomaten für alle

Vorlesungen gibt. Da Punkte durch verschiedene Aufgabenarten gesammelt werden können und sollen, ist das ein Vorschlag, deren Umsetzung denkbar wäre. Ein Student wünscht sich Aufgaben, die den Schwierigkeitsgrad der Prüfung übersteigen. Da die einzelnen Aufgaben für die verschiedenen Vorlesungen jeweils angefertigt werden müssen, könnten diese auch mit verschiedenen Schwierigkeitsgraden konzipiert werden. Für die einzelnen Übungen werden auch Musterlösungen und Kontrolle, sowie Lösungsvorschläge oder Hinweise gewünscht. Ob und in wie weit so etwas in Übungen integriert werden kann, müssen die Mitarbeiter und Professoren für die einzelnen Vorlesungen entscheiden. Auch dafür, ob durch das Erfüllen der Aufgaben Zusatzpunkte für die Prüfungen gesammelt werden können, bedarf es einer Einzelfallentscheidung für jede Vorlesung. Ein Vorteil von Zusatzpunkten ist die Motivation, die Studenten dadurch bekommen. Auch ein möglichst geringer Zeitaufwand, wie von einem Studenten gewünscht, trägt dazu bei, dass die Motivation nicht geschmälert wird. Weitere Anmerkungen richten sich dahin, dass auf den Datenschutz geachtet werden soll. Bei dem Match-Making müssen jedoch den Studenten die Namen der Partner bekannt sein, ebenso können Tutoren oder andere Mitarbeiter so gezielt den schwachen Studenten auch offline helfen. Schließlich wurde noch angemerkt, dass es schon viele verschiedene Websites oder Hilfsangebote gibt und eine Vereinheitlichung wünschenswert wäre. Andernfalls könnten Studenten dieses Programm als überflüssig empfinden und nicht nutzen. So wie ein Kommentar zum Auditorium zeigt: ”Auditorium, wer nutzt das schon!”.

4.2 Anforderungsanalyse

Die Wünsche der Studenten betreffen kaum das Match-Making an sich, sondern stellen eher allgemeine Anforderungen an das Programm dar. Da das Match-Making jedoch das Kernstück des Programms werden soll, wird hier zunächst auf die Anforderungen daran eingegangen und andere Forderungen an das Programm werden danach diskutiert. In Kapitel 2.1 wurden schon einige Anforderungen an das Match-Making zwischen Studenten aufgestellt und anschließend verschiedene Protokolle und Algorithmen daraufhin untersucht, ob sie diese erfüllen.

So soll das Match-Making grundsätzlich den besten Student dem schlechtestem Studenten zuordnen. Dafür ist es notwendig den bisherigen Leistungsstands eines jeden einzelnen Studenten je Lehreinheit zu ermitteln und einen Wert zuzuordnen, so dass die Studenten miteinander verglichen werden können. Da sich eine Vorlesung aber in den meisten Fällen lediglich über ein Semester erstreckt und das Match-Making von Anfang an aktiv sein soll, muss der Leistungsstand der Studenten schnell und einfach ermittelbar sein. Dafür wären zum Beispiel kleine Aufgaben während der Vorlesung geeignet, bei denen es für richtige Lösungen Punkte gibt. Die Abstraktion von einer erreichten Punktzahl zu einer Note wäre für das Match-Making hinderlich, da über viele Aufgaben hinweg, die Noten gemittelt werden müssen, während die erreichten Punkte aufsummiert werden können. Somit kann durch die Nutzung von Punkten allein eine stärkere Differenzierung der Leistungen

der Studenten sichtbar gemacht werden. Mit den Punkten kann eine Rangliste gebildet werden, ähnlich wie beim Match-Making in Computerspielen. Mit Hilfe so eine Rangliste können verschiedene Algorithmen Studenten je nach Anwendungsfall einander zuordnen oder Gruppen bilden. Weitere Informationen über die Studenten sind dafür nicht nötig. Zusammenfassend muss das Match-Making folgendes können:

- Das Match-Making soll auf Grundlage einer Rangliste vorgenommen werden.
- Die Punkte für die Rangliste sollen im Rahmen der Vorlesung oder des Seminars erarbeitet werden können.
- Je nach Anwendungsfall, können verschiedene Algorithmen für das Match-Making genutzt werden. Dabei wurden das kontrollierende Matching, das sofortige Matching und das Gruppen-Matching vorgestellt.

Zusätzlich und optional zu den oben genannten Anforderungen könnten zeitintensive Aufgaben mit Meilensteinen versehen werden, um den Fortschritt festzuhalten und fertige Studenten immer den langsamsten zuzuordnen. Ebenso würde es die Motivation und das Wissen der Studenten steigern, Hilfsanfragen und -angebote einstellen zu können und danach gematcht zu werden.

Der wichtigste Punkt bei den allgemeinen Anforderungen an das Programm ist, wie in der Kurzbefragung bestimmt wurde, die Rückmeldung über den Lernfortschritt. Dieser kann leicht durch die schon benötigten Punkte dargestellt werden. Zur besseren Veranschaulichung und als Teil von Gamification können aus diesen Punkten Level berechnet werden. Dadurch ist auf den einen Blick der Fortschritt, sowie ein ungefährer Wissensstand ersichtlich. Weiterhin wünschen sich die Studenten einen möglichst geringen zusätzlichen Zeitaufwand. Wenn die Aufgaben, wie zum Beispiel Multiple-Choice-Tests, in Vorlesungen gestellt und beantwortet werden müssen, verringert sich der Zeitaufwand deutlich. Somit lassen sich die Studenten besser dazu motivieren, das Programm zu nutzen, da sie mit wenig Zeitaufwand viel erreichen wollen. Auch die Bereitstellung von Lösungen oder Hinweisen zu Aufgaben kann die Nutzungsbereitschaft der Studenten erhöhen. Denn dadurch erhalten sie im Gegensatz zum Auditorium zum Beispiel sofortige Rückmeldung über mögliche Fehler. Dies kann jedoch nur von den einzelnen Aufgabenstellern bereitgestellt werden, ebenso der Wunsch nach praktischen Aufgaben, ähnlich zum Praktomat.

Für weitere Gamification-Elemente wie Abzeichen konnte sich in der Kurzbefragung keine Mehrheit begeistern. Solche Elemente können jedoch auch später hinzugefügt werden, falls sie dann gewünscht sind. Abzüglich dessen werden folgende Anforderungen allgemein an das Programm gestellt:

- Die Rückmeldung über den Lernfortschritt soll durch Punkte und/oder Level erfolgen.
- Ein geringer zusätzlicher Zeitaufwand erhöht die Motivation, das Programm zu nutzen.

- Lösungen und/oder Hinweise zu Aufgaben bieten sofortige Rückmeldung für hilfsbedürftige Studenten. Dies muss durch die jeweiligen Aufgabensteller bedacht werden.
- Praktische Aufgaben, wie das Programmieren, steigern das Verständnis für den Lehrstoff. Es sollten nicht nur theoretische Aufgaben gestellt werden.

4.3 Konzeption

Aus der Anforderungsanalyse wird ersichtlich, dass verschiedene Algorithmen für das Match-Making nötig sind, sie jedoch alle auf derselben Rangliste ausgeführt werden. Dabei hat jede Vorlesung ihre eigene Rangliste von Studenten und jeder Student benötigt Informationen über seinen Punktestand in jeder Vorlesung und die Studenten, die das Match-Making für ihn ergeben hat. Die Punkte wiederum sollen durch Aufgaben in den jeweiligen Vorlesungen erworben werden können. In einer objektorientierten Programmiersprache werden für solch ein Programm die Klassen Student, Aufgabe und Lehrveranstaltung benötigt. Sowie eine Klasse für das Match-Making, welche die verschiedenen Algorithmen beinhaltet. Abbildung 4.2 zeigt die vorhandenen Klassen in einem UML-Klassendiagramm und deren Beziehungen untereinander. Die Aufgaben der einzelnen Klassen werden im folgenden kurz beschrieben.

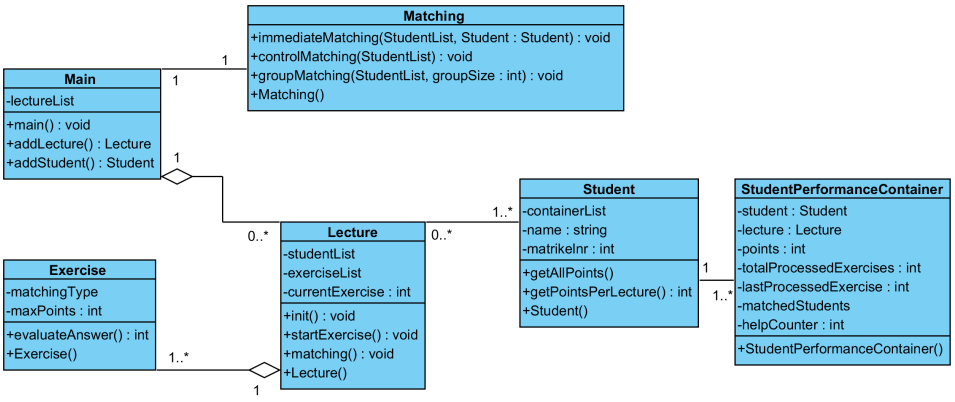


Abbildung 4.2: UML-Klassendiagramm zur Konzeption

Student: Diese Klasse besitzt Standardinformationen über den Studenten, wie z. B. Name, Matrikelnummer, Semester, etc... Ein Student kann mehrere Lehrveranstaltungen in einem Semester besuchen. Alle Daten zu den einzelnen Lehrveranstaltungen werden in einem extra Container gespeichert. Der Student besitzt eine Liste dieser Container.

Lecture: Diese Klasse soll die Vorlesung oder das Seminar repräsentieren. So eine Lehrveranstaltung ist für jeweils ein Semester gültig und kann eine weiterführende Lehrveranstaltung oder selber Grundlage oder Teil einer Lehrveranstaltungsreihe sein. Daher sollen Punkte aus vorhergehenden Lehrveranstaltungen oder Klausuren teilweise übernommen werden und somit als erste Ordnung der Studenten für das Matching dienen können.

Exercise: Eine Aufgabe kann sehr vielfältig sein: z. B. Multiple Choice Tests, Programmieraufgaben, etc... Es soll jedoch immer ein Matchingtyp angegeben werden, der bestimmt, welcher Matchingalgorithmus für die jeweilige Aufgabe genutzt werden soll. Auch die maximal erreichbare Punktzahl, die ein Student für die Erfüllung der Aufgabe bekommen kann, muss festgelegt werden. Des Weiteren müssen die Antworten der Studenten ausgewertet werden können. Das kann für jeden Aufgabentyp anders gehandhabt werden.

StudentPerformanceContainer: Dieser Container wird für jeweils einen Studenten und eine Lehrveranstaltung erstellt und soll sämtliche Daten, die ein Student zu einer Lehrveranstaltung hat, speichern. So werden hier die erreichten Punkte und die Anzahl der bearbeiteten Aufgaben gespeichert. Zusätzlich wird gezählt, wie oft der Student einem anderen geholfen hat. Wenn gerade ein Matching aktiv ist, der Student also einem oder mehreren anderen Studenten zugeordnet ist, werden diese in einer Liste gespeichert.

Matching: Diese Klasse stellt verschiedene Matchingalgorithmen zur Verfügung, da Aufgaben unterschiedliche Anforderung an eine Zuordnung zwischen Studenten haben können. Der passende Algorithmus soll in der Exercise-Klasse oder Lecture-Klasse, falls ein Matching ohne konkrete Aufgabe durchgeführt werden soll, gewählt werden.

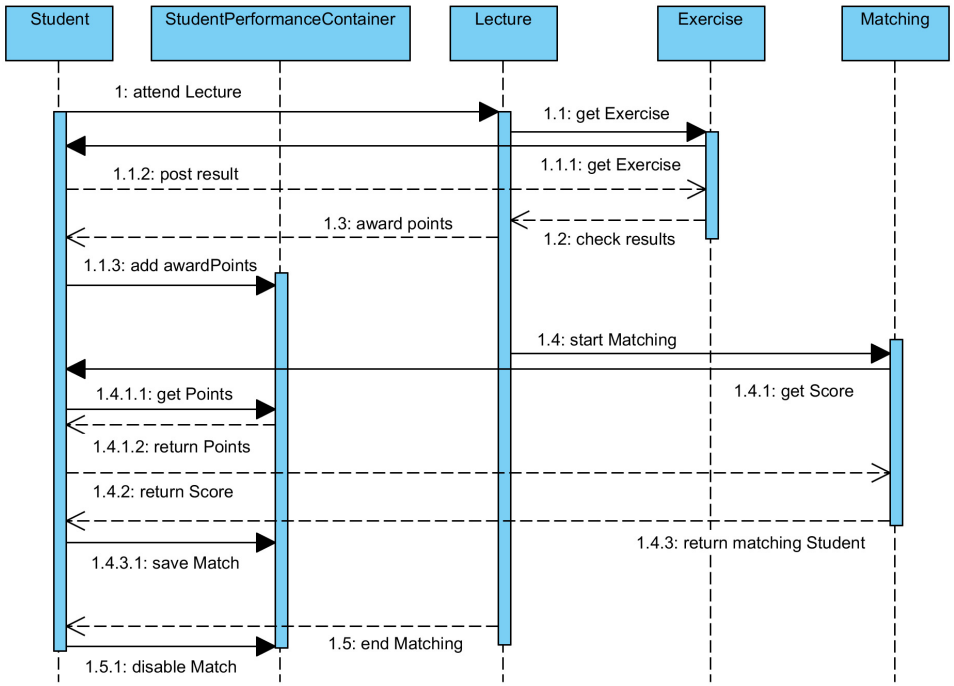


Abbildung 4.3: Sequenzdiagramm für das Bearbeiten von Aufgaben und anschließendes Matching

In Abbildung 4.3 sind 2 wichtige Abläufe zwischen den einzelnen Klassen dargestellt. Im ersten Abschnitt ist die Zuteilung einer Aufgabe mit abschließender Punktevergabe zu sehen. Dazu muss sich der *Student* zunächst bei der *Lecture* als aktiv anmelden. Dies ist nötig, da Studenten auf Grund von Krankheit oder anderem verhindert sein können und

somit weder die Aufgaben bearbeiten können, noch für ein anschließendes Matching zur Verfügung stehen würden.

Nachdem der *Student* bei der *Lecture* angemeldet ist, holt sich selbige die aktuelle Aufgabe (*Exercise*). Diese wird dem *Student* übergeben, der sie bearbeitet und das Ergebnis zurückgibt. Dieses wird von *Exercise* überprüft und der *Lecture* wird die erreichte Punktzahl übergeben. Das geschieht, weil somit die *Lecture* einen Studenten, sobald er eine Aufgabe beendet hat, mit einem noch nicht fertigen Studenten matchen lassen kann, wenn dies von der Aufgabe her gewünscht ist. Die Bearbeitung der Aufgabe schließt damit ab, dass der *Student* die verdienten Punkte übermittelt bekommt und diese in dem *StudentPerfomanceContainer* für die jeweilige *Lecture* zu den schon erworbenen Punkten addiert und speichert.

Das Matchmaking zwischen den Studenten kann je nach *Exercise* zu verschiedenen Zeitpunkten erfolgen. So kann für Gruppenaufgaben das Matchmaking vor der Aufgabenbearbeitung stattfinden, während die Studenten bei Programmieraufgaben sofort nach der Beendigung ihrer Aufgabe einem anderen Studenten zugeordnet werden, um sofort helfen zu können. In der Abbildung 4.3 ist eine dritte Möglichkeit des Zeitpunktes für das Matchmaking dargestellt. Für die gegenseitige Kontrolle der Aufgaben werden die Studenten einander erst dann zugeordnet, wenn alle die *Exercise* bearbeitet haben. Die *Lecture* startet zu dem entsprechenden Zeitpunkt das *Matching* und wählt den zu verwenden Algorithmus aus. Das *Matching* benötigt für die Berechnung die Punkte der Studenten. Über *Student* werden diese aus dem *StudentPerformanceContainer* geholt. Abschließend werden dem *Student* sein Match, bestehend aus einem oder mehreren Studenten, übergeben. Das Match wird vom *Student* im *StudentPerformanceContainer* gespeichert bis die *Lecture* das Ende des Matchmakings verkündet.

4.4 Befragung zur Konzeption

Um heraus zu finden, ob die Konzeption des Programms den Anforderungen gerecht wird oder ob es Verbesserungs- und Erweiterungsvorschläge gibt, wurde ein Expertenbefragung mit ukrainischen Studenten durchgeführt. Diese waren für ein paar Wochen zum Austausch an der TU Dresden und studieren Informatik. Somit besteht keinerlei Überschneidung mit den Studenten, die schon an der Kurzbefragung teilgenommen hatten. Die Ukrainer sind der Konzeption gegenüber also unvoreingenommen und müssen somit ihre spontanen Gedanken dazu äußern. Des Weiteren sind sie auch gleichzeitig die Zielgruppe des Programms: Studenten und da sie Informatik studieren, haben sie das nötige Wissen, um die Konzeption zu bewerten. Dafür wurden 3 Aufgaben erstellt, die sie jeweils in einer Gruppe lösen sollten.

Gruppe 1.
Die erste Gruppe wurde zum Match-Making befragt. Dazu bekamen sie die Kurzbeschreibung aus der Kurzbefragung und das UML- und Sequenzdiagramm. Die Studenten sollten beschreiben, was sie von einem Match-Making-Sestem erwarten, das Studenten einander

so zuordnet, dass die Leistungsstarken den Leistungsschwachen helfen können. Anschließend sollten sie sich Gedanken darüber machen, welche Arten des Matchings genutzt werden können. Somit sollten Erwartungen und Ergänzungen zum Match-Making erfasst werden.

Die Studenten erwarten von dem Match-Making einerseits die Erhöhung des durchschnittlichen Wissensstandes und das Sammeln von Erfahrungen durch Teamwork. Um den leistungsstarken Studenten einen Anreiz für gute Erklärungen zu geben, wird vorgeschlagen, dass die hilfesuchenden Studenten Abzeichen für die Helfenden vergeben können, wenn diese das Wissen gut vermitteln. Studenten mit vielen Abzeichen könnten dadurch bei Hilfsanfragen eher gematcht werden. Das würde dem Match-Making eine zusätzliche Dimension geben. Zunächst sollten jedoch die einfachen Algorithmen zum Match-Making getesteten werden. Die Idee mit den Abzeichen kann in einer späteren Version des Programms umgesetzt werden. Die Erhöhung des Wissensstandes, sowie die Erfahrungen mit Teamwork, sind Ziele des Match-Makings und sollten ebenfalls durch einen Test des kompletten Programms überprüft werden.

Für diese Arbeit besonders wichtig sind die Strategien des Match-Makings. Die Befragten haben sich dabei allerdings hauptsächlich auf das Matching zu Gruppen fokussiert. Eine Eins-zu-Eins-Zuordnung wurde von ihnen nicht bedacht. So wurden drei Arten von Match-Making vorgeschlagen. Die erste Möglichkeit wäre, dass der beste Student nach dem Test zu einem Thema, eine Erklärung oder einen Kurzvortrag darüber hält. Anschließend folgt eine neue Bewertung der Studenten. Das hat jedoch den Nachteil, dass viel zusätzliche Zeit benötigt wird und die Erklärung des Lehrstoffs für alle Studenten nicht auf die Bedürfnisse und Fragen der Leistungsschwachen eingeht. Des Weiteren wurde der zu vermittelnde Lehrstoff zu dem Zeitpunkt, an dem der Student einen Vortrag hält, schon von Vorlesenden oder Tutoren erklärt. Der Nutzen für die schwächeren Studenten ist dadurch fraglich. Die zweite vorgeschlagene Variante des Match-Makings separiert zunächst die Studenten in eine Gruppe der Leistungsstarken und eine der Leistungsschwachen, mit zum Beispiel 25 % in der ersteren und die restlichen 75 % bilden die schwache Gruppe. Anschließend sollen daraus kleine Gruppen wie folgt gebildet werden: einem Studenten aus der guten Gruppe werden drei Studenten aus der Gruppe der Leistungsschwachen zugeordnet. Dabei ist jedoch nicht sichergestellt, dass der Beste mit dem Schlechtesten zusammen gruppiert wird. Das führt dazu das im schlimmsten Fall der schlechteste Student aus der Gruppe der Leistungsstarken mit den drei schlechtesten Studenten in eine Gruppe kommt. Das ist eine ungemeine Belastung für diesen Studenten und kann dazu führen, dass er überfordert wird und selber in seinen Leistungen stark nachlässt. Auch für seine Gruppenpartner kann diese Art Match-Making unvorteilhaft sein. So sind vielleicht alle auf die Erklärungen das Starken angewiesen, können sich gegenseitig aber nicht helfen und bräuchten verschiedene Erklärungsansätze für ihr Verständnis. Damit hat der starke Student wenig Zeit für jeden Einzelnen der Leistungsschwachen. Eine letzter Vorschlag für das Match-Making für Gruppen ist die Gruppierung von Studenten mit ähnlichem Leistungsstand. Das heißt zum Beispiel die besten 5 Studenten sind in einer Gruppe und die schlechtesten 5 in einer anderen. Die Idee dahinter war, dass selbst von den leistungsschwachen Studenten jeder

zumindest einen teilweisen Einblick in die Materie hat und sie sich daher gegenseitig helfen können. Bei so komplexen und neuen Themen, wie sie durch Vorlesungen im Studium gelehrt werden, kann jedoch bezweifelt werden, dass in solchen Gruppen genügend Wissen und Verständnis für den Lehrstoff herrscht, damit sich auch die schwächsten gegenseitig helfen können. Diese Art des Match-Makings ist daher abzulehnen. Insgesamt zeigt sich, dass vor allem das Match-Making für Gruppen komplex zu sein scheint und ein Algorithmus dafür zum einen den durchschnittlichen Wissensstand der Gruppen einheitlich halten muss und zum anderen einzelne Studenten nicht überfordern darf.

Gruppe 2.
Die zweite Gruppe sollte zum UML-Diagramm befragt werden. Dafür bekamen sie die Kurzbeschreibung der Kurzbefragung, sowie das UML-Diagramm mitsamt der Beschreibung der einzelnen Klassen. Ihre Aufgaben waren, zu bewerten, ob das Klassendiagramm den Anforderungen an das Programm genügt und ob es erweiterbar für zukünftige Verbesserungen ist. Leider verfügten die Studenten in der zweiten Gruppe nicht über ausreichende UML-Kenntnisse und konnten somit die Aufgaben nicht lösen. Die Gruppe wurde daher aufgelöst und die Studenten wurden einer der anderen Gruppen zugeordnet.

Gruppe 3.
In der dritten Gruppe der Befragten sollten diese die Rolle der Studenten, die das Programm benutzen, untersuchen. Dazu bekamen sie die Kurzbeschreibung der Kurzbefragung und das Sequenzdiagramm mitsamt der Beschreibung. Sie sollten diskutieren, was sie als Studenten gerne sehen würden (zum Beispiel Punkte, das Level für jede *Lecture*) und wodurch sie so ein Programm öfter nutzen würden.

Als Studenten wollen die Befragten vordergründig eine Liste mit besuchten Lehrveranstaltungen und den dazugehörigen Punkten und/oder das Level sehen. Als Detailansicht wünschen sie sich für jedes Fach eine Tabelle mit den Aufgaben und den verdienten Punkten, nach dem Datum der Aufgabe sortiert, sowie die gesamte bisher erreichte Punktzahl für die entsprechende Lehrveranstaltung. Die Datenstrukturen, die dafür nötig sind, wurden schon zum Großteil im *StudentPerformanceContainer* konzipiert, zusätzlich könnte eine Map mit der Aufgabennummer als Schlüssel und der erreichten Punktzahl als Wert eingefügt werden. Des Weiteren möchten die Befragten Punkte, die durch Hilfe für andere Studenten verdient wurden, extra aufgeführt sehen. Diese können in einer zusätzlichen Variable gespeichert werden. Für den gesamten Überblick finden die Studenten auch einen Stundenplan hilfreich, der zum Teil mit den schon vorhandenen Informationen bereitgestellt werden könnte. Da Übungszeiten für jeden Studenten unterschiedlich sind, müssten sie entweder von Hand hinzugefügt oder vernachlässigt werden. Dies geht jedoch über die Zielstellung dieser Arbeit hinaus.

Die Lehrveranstaltungen betreffend wünschen sich die Befragten die jeweiligen Materialien dazu online oder als Links integriert. Auch Videos von Vorlesungen und zusätzliche Ressourcen, wie zum Beispiel Papers oder Links fänden die Studenten hilfreich. Ebenso wollen sie ein Rating für die Lehrveranstaltungen sowie Noten oder Punkte an das Lehrpersonal vergeben. Es ist jedoch fraglich, ob die Bewertung durch die Studenten immer

angemessen ist oder hilfreich für andere Studenten. So können Studenten vorschnell eine negative Wertung abgeben, wenn sie über eine Aufgabe oder ähnliches verärgert sind. Viele Lehrveranstaltungen sind außerdem Pflicht und müssen absolviert werden. Erst im Verlauf des Studium mit der Spezialisierung können Fächer gewählt werden, eine einfache Note ist aber für die Entscheidung, welche Lehrveranstaltungen besucht werden soll, nicht ausreichend.

Für die Aufgaben wünschen sich die Studenten neben einem Titel auch eine Beschreibung und die Angabe des Schwierigkeitsgrades der Aufgabe, sowie Lösungen. Letztere sollten allerdings erst nach Bearbeitung und Vergabe der Punkte angezeigt werden. Zusätzlich könnte eine Liste mit Referenzen und weiteren Ressourcen angegeben werden, die je nach Schwächen (keine Punkte in den entsprechenden Aufgabenteilen) der Studenten zusammengestellt wird. Wichtig war den Befragten auch, dass sie sich Ziele setzen können: wie viele Punkte oder welches Level sie im Verlauf des Semesters erreichen wollen. So eine Zielsetzung kann motivierend sein, aber sobald das Ziel erreicht ist, besteht die Möglichkeit, dass Studenten nicht weiter mit dem Programm arbeiten und auch für das Match-Making nicht mehr zur Verfügung stehen. Es wurde auch eine Liste von Studenten gewünscht, die an derselben Aufgabe arbeiten, um sich somit in Echtzeit mit Anderen austauschen zu können. Da die zu bearbeitenden Aufgaben zum Teil allein gelöst werden sollen, damit keine Verfälschung der Leistung eintritt, ist der Vorschlag in diesen Fällen abzulehnen. Für die anderen Aufgaben, wie längere Programmieraufgaben, kann ein sofortiges Matching ausgeführt werden. Das ordnet Studenten, welche die Aufgabe komplett bearbeitet haben, den schlechtesten Studenten zu, die noch keinen Matching-Partner haben. Außerdem werden alle Studenten in der *Lecture* geführt. Sollte eine solche Liste zukünftig benötigt werden, kann sie daraus abgeleitet werden.

Bei der Frage danach, was Studenten so ein Programm öfter benutzen lässt, war den Befragten ein gutes User Interface mit Abstand am wichtigsten. Die graphische Oberfläche des Programms ist das erste mit dem die Studenten in Kontakt kommen. Eine schlechte GUI würde mit Sicherheit viele der Studenten von der Verwendung des Programms abhalten. Gleiches gilt für eine zu komplizierte Anwendung. Die Funktionen des Programmes sollten also einfach und leicht verständlich sein. Zusätzlich regten die Befragten die Nutzung von Gamification-Elementen an, wie zum Beispiel eine Animation bei voller Punktzahl für eine absolvierte Aufgabe, die als Belohnung dient und somit motivieren soll. Auch extra Punkte, die auf verschiedene Arten wie Mitarbeit, Hilfestellung, etc. neben den normalen Punkten für die Aufgaben verdient werden könnten, bringen weitere Motivation. Punkte als Belohnung für eine Klausur wurden ebenfalls gewünscht. Da eine *Lecture* jedoch zumeist mit einer Klausur endet, ist dies nur bei Vorlesungsreihen, bestehend aus mehreren Vorlesungen in aufeinanderfolgenden Semestern sinnvoll. Dabei soll allerdings schon durch die gesammelten Punkte der vorigen Vorlesung oder durch die Leistung in der Klausur an sich, eine Punktzahl berechnet werden, die als Anfangspunkte in die folgende Vorlesung einfließen. Somit sind solche Punktebelohnungen nicht nützlich. Neben diesen Gamification-Elementen ist den Befragten auch die Möglichkeit zur Kommunikation mit den Lehrpersonen und Kommilitonen wichtig. Diese sollte direkt und ohne Verzögerung

erfolgen, ähnlich wie in einem Chatroom oder einer Lobby in Computerspielen. Das Warten auf Antworten in Foren oder via Emails empfinden die Meisten als zu zeitaufwendig und nicht interaktiv genug, da sich aus einer gestellten Frage leicht weitere Folgefragen ableiten lassen. Auch Aktualität ist für Studenten wichtig. Das betrifft sowohl Lehrveranstaltungen und Übungsergebnisse, wie auch Wissenschaft, Innovationen und Forschung im Allgemeinen. Durch das Bereitstellen aktueller Artikel zu diesen Themenfeldern kann das Interesse daran geweckt oder erhalten werden. Zusätzlich würden solche News dazu führen, dass Studenten das Programm intensiver nutzen, wenn sie sich darüber informieren. Gegensätzlich dazu wurden von den Befragten weitere verteilte Quellen identifiziert. Eine zu große Verteilung von Inhalten über Webseiten, Foren, etc. hinweg verringert die Nutzung jeder einzelnen Quelle. Die Studenten würden sich also auf wenige Quellen beschränken und das zu entwickelnde Programm als neuartige Möglichkeit, würden sie nicht mehr in Betracht ziehen. Wohingegen eine gute Nutzung des Programms unter den Studenten und eine mögliche Verbreitung auch an anderen Universitäten die Nutzung dessen erhöhen würde. Schlussendlich fordern die Befragten auch das Sammeln und Umsetzen von Nutzer-Feedback, um die Verwendbarkeit weiter zu verbessern und eventuell weitere Bedürfnisse an das Programm zu identifizieren. All das sind Dinge, die für die Frontend-Entwicklung berücksichtigt werden sollten, jedoch in dieser Arbeit zum Match-Making nicht berücksichtigt werden können.

5 Implementierung

Das im vorherigen Kapitel konzipierte Programm wurde mit wenigen Änderungen nach der zweiten Befragung prototypisch in Java implementiert. Der Fokus lag dabei auf der Implementierung der Algorithmen für das Match-Making, also der Klasse *Matching*. Diese Algorithmen werden im folgenden vorgestellt. Davor werden die anderen Klassen und die Änderungen erklärt.

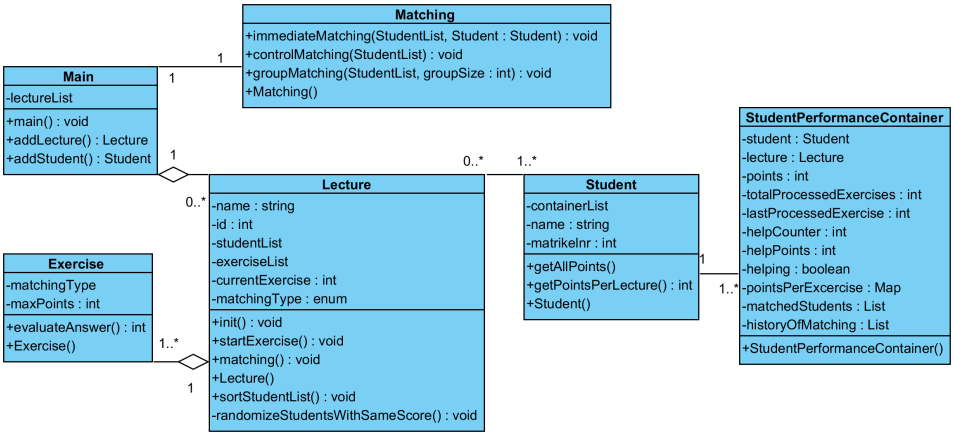


Abbildung 5.1: Erweitertes UML-Klassendiagramm mit wichtigen Attribute und Funktionen ohne Getter und Setter

Abbildung 5.1 zeigt eine Erweiterung des in der Konzeption vorgestellten Programms und somit das Grundgerüst, das implementiert wurde. Die folgende Beschreibung der Klassen bezieht sich auf diese Abbildung. Um die Algorithmen zum Match-Making zu testen war es nötig eine *Lecture* anlegen zu können, die eine Rangliste der Studenten beinhaltet, die *studentList*. Des Weiteren besteht die *Lecture* neben dem Namen und einer Id aus einer *excerciseList*, die sämtliche *Excercises* der *Lecture* enthält, einem Wert für die Anzeige der aktuellen *Excercise* und einem Enum, *matchingType*, in dem die möglichen Arten des Matchings deklariert sind. Die Klasse *Excercise* selbst ist für die Tests der Match-Making-Algorithmen lediglich als Dummy nötig, um den Studenten Punkte zuordnen zu können und enthält daher noch keine Funktionalität. Mit Hilfe des jeweils in der *Excercise* oder *Lecture* gewählten *matchingType* wird in der Methode *matching()* der entsprechende Match-Making-Algorithmus gestartet. Da diese Algorithmen auf einer Rangliste der Studenten, sortiert nach den verdienten Punkten, ausgeführt werden, ist auch Funktion zum Sortieren der Studenten in der *studentList* nach ihrem Score nötig. Damit bei gleicher Punktzahl der

Zufall entscheidet, wer in der Liste auf welchem Platz steht, wird nach jeder Ausführung von *sortStudentList()* auch *randomizeStudentsWithSameScore()*, eine Funktion zur zufälligen Umordnung von Studenten mit gleichem Score in einer sortierten Rangliste, ausgeführt. Weitere Funktionen dieser Klasse sind neben den nicht näher betrachteten Gettern und Settern *startExcercise()*, um eine Aufgabe zu beginnen und die *init()*-Funktion, die genutzt werden soll, falls der *Lecture* eine andere Vorlesung zu Grunde legt, um den Punktestand übertragen zu können. Dadurch können die Studenten schon zu Beginn der Vorlesung ihrer Leistung entsprechend in die Rangliste sortiert werden.

Für die Tests der Algorithmen ist es weiterhin nötig Studenten anlegen zu können, sie eine *Lecture* besuchen zu lassen und Informationen darüber zugänglich speichern zu können. Dafür wurde die Klasse *Student* angelegt. Diese speichert neben persönlichen Informationen über den Studenten auch eine Liste, die *containerList*, von *StudentPerformanceContainern*. Ein solcher Container enthält alle Daten zum persönlichen Abschneiden in einer besuchten *Lecture*. Über den *Student* werden diese Daten zugänglich gemacht. So gibt es Funktionen wie *getPointsPerLecture()*, die den erreichten Punktwert zu einer besuchten *Lecture* zurückgeben, und ähnliche Getter und Setter. Die Daten, die im *StudentPerformanceContainer* abgelegt werden, sollen einerseits dem Studenten und Betreuern Rückmeldung geben und andererseits dem Match-Making dienen und wurden durch einige Vorschläge aus der Befragung zur Konzeption erweitert. So werden neben dem Score – gespeichert in *points* – auch die erreichten Punkte für jede einzelne Aufgabe in *pointsPerExcercise* gespeichert, sowie die durch das Helfen verdienten Punkte in *helpPoints*. Dadurch erhalten die Studenten eine detaillierte Rückmeldung über ihre erbrachten Leistungen. Der *helpCounter* zeigt ihnen an, wie oft sie Anderen geholfen haben. Die Werte *totalProcessedExcercises* und *lastProcessedExcercise* zeigen die insgesamt und zuletzt bearbeiteten Aufgaben und können als Information für Betreuer und Vorlesende betrachtet werden oder für eine Anpassung der Punktevergabe durch Aging herangezogen werden. Eine Liste mit den aktuell dem *Student* durch das Match-Making zugeordneten anderen Studenten ist *matchedStudents*. Alle ehemaligen Matches werden in der Liste *historyOfMatching* abgelegt.

Die Algorithmen für das Match-Making wurden in der Klasse *Matching* angelegt und können theoretisch von allen anderen Klassen verwendet werden. Jedoch werden sie vor allem in der Klasse *Lecture* benötigt und könnten teilweise in der *Excercise* genutzt werden. Die Klasse *Matching* enthält die drei Algorithmen, die in Kapitel 3 beschrieben wurden. Das kontrollierende Match-Making soll Paare bilden, die sich gegenseitig nach Abschluss einer *Excercise* kontrollieren und wird hauptsächlich in Vorlesungen Anwendung finden, kann jedoch auch in Seminaren, Übungen und Praktika verwendet werden. Der zweite Match-Making-Algorithmus ist das sofortige Match-Making. Es ist nützlich für praktische Aufgaben, Seminare und überall, wo programmiert wird, da Studenten, die mit einer Aufgabe fertig sind sofort Anderen helfen können. Zuletzt folgt das Gruppen-Matching, welches Gruppen für verschiedene Aufgaben, z. B. für ein Programmierpraktikum oder bestimmte Übungen, zusammenstellen kann. Auch Lerngruppen könnten damit gebildet werden. Die Funktionsweisen dieser Algorithmen werden im Folgenden genauer beschrieben.

1. control-Matching (Pair-Matching)

Pseudocode 5.1: 1. control-Matching (Pair-Matching)

```
controlMatching(List studentList, Lecture lecture){
 sort(studentList);
 while(studentList is not empty){
  if(3 or less students in studentList){
   for(each student in studentList){
        addMatch(student, List of other students, lecture);
        return;
   }
  }
  fetch first and last student from studentList;
  addMatch(first student, last student, lecture);
  removeFromStudentList(first student);
  removeFromStudentList(last student);
 }
}
```

Das kontrollierende Match-Making oder auch Paarbildung, wird auf einer Rangliste der Studenten durchgeführt. Diese Liste muss sortiert sein. Auf Grund des Aufbaus des Programms muss ebenfalls die *Lecture* bekannt sein, da ein Student in zwei verschiedenen *Lectures* auch zwei verschiedene Matches haben kann. Ist die übergebene Liste leer oder beinhaltet nur einen Studenten kann das Match-Making nicht durchgeführt werden und der Algorithmus bricht ab. Andernfalls werden von der sortierten Listen in jedem Schleifendurchlauf der erste und der letzte Student genommen, einander zugeordnet und aus der Liste entfernt. Wenn nur noch zwei oder drei Studenten in der Liste stehen, werden diese einander zugeordnet, sodass das letzte Match bei einer ungeraden Anzahl an Studenten eine Gruppe von drei Studenten bildet. Die jeweils zugeordneten Partner werden der Klasse *Student* übergeben und im zur *Lecture* gehörenden *StudentPerformanceContainer* gespeichert.

2. immediate Matching

Pseudocode 5.2: 2. immediate Matching

```
immediateMatching(List studentList, Student helpingStudent,
Lecture lecture){
 sort(studentList);
 if(helpingStudent is already matched){
  return matching not possible;
 }
 for(each student in studentList in ascending order){
  if(student ist not matched && student is not helpingStudent){
   addMatch(student, helpingStudent, lecture);
   return match found;
  }
 }
}
```

Das sofortige Match-Making wird ebenfalls auf einer sortierten Rangliste ausgeführt. Diese muss aufsteigend sortiert sein. Zusätzlich wird der zuzuordnende Student benötigt. Falls dieser schon ein Match hat, wird der Algorithmus abgebrochen. Andernfalls wird in der Rangliste beginnend beim schlechtesten Studenten der erste Student ohne Match gesucht. Das heißt der schlechteste Student, dem noch nicht durch einen anderen Studenten geholfen wird. Dieser bildet abschließend mit dem übergebenen zuzuordnenden Studenten ein Match, welches wie auch beim kontrollierenden Match-Making für die Studenten im zur entsprechenden *Lecture* gehörenden *StudentPerformanceContainer* gespeichert wird. Falls alle Studenten außer dem Zuzuordnenden schon ein Match haben, überprüft der Algorithmus das und beendet sich, ohne dem übergebenen Studenten einen anderen zuordnen zu können.

3. Group-Matching

Pseudocode 5.3: 3. Group-Matching

```
groupMatching(List studentList, int groupsize,
boolean roundup, Lecture lecture){
 sort(studentList);
 if(groupsize too big, too small or not good){
  return not a good grousize;
 }
 int size = size(studentList);
 int leftoverStudents = size % groupsize;
 int numberOfFullGroups = size / groupsize;
 boolean reduceSize = false;

 for(int i = numberOfFullGroups; i >= 0; i--){
  if(studentList is empty) return;
  List group;
  if(i == 0){
   group = studentList;
   matchStudentsInGroup(group, lecture);
  }
  int help = groupsize/2;
  int specificGroupSize = groupsize - help;
  if(!roundup && leftoverStudents between 0 and groupsize){
   ... this group contains one student less...
   ... than groupsize...
   ... recalculate specificGroupSize...
  }

  for(int m = 0; m < (specificGroupSize); m++){
   if(m < help){
    int index = m * i - m;
    group.add(studentList.remove(index));
    group.add(studentList.remove(
                    (studentList.size() - 1) - index));
   }else{
```

```
34      group.add(studentList.remove(
35                          (studentList.size()-1)/2));
36     }
37    }
38    if(roundup && leftoverStudents > 0){
39     group.add(studentList.remove(
40                         (studentList.size()-1)/2));
41     leftoverStudents--;
42    }
43    matchStudentsInGroup(group, lecture);
44   }
45  }
46
47  matchStudentsInGroup(List group, Lecture lecture){
48   for(each student in group){
49     addMatch(student, List of other students in group, lecture);
50   }
51  }
```

Der dritte Match-Making-Algorithmus, der implementiert wurde, ordnet die Studenten einer Rangliste auf Grund ihrer Leistungen in Gruppen. Neben einer aufsteigend sortierten Rangliste der Studenten benötigt der Algorithmus auch die Gruppengröße, also wie viele Studenten eine Gruppe bilden sollen, die zugehörige *Lecture* und einen Wert der anzeigt, ob auf- oder abgerundet werden soll. Dieser Wert wird dann benötigt, wenn bei der Gruppeneinteilung ein oder mehrere Studenten übrig sind, die keine Gruppe mehr in der geforderten Gruppengröße bilden können. Ist der Wert *roundup* wahr, so werden diese restlichen Studenten gleichmäßig auf andere Gruppen verteilt. Ist der Wert hingegen nicht wahr, werden so lange Gruppen mit einem Studenten weniger als von der Gruppengröße gefordert gebildet, bis die Zahl der noch nicht gruppierten Studenten durch die Gruppengröße ohne Rest teilbar ist. Durch die Verwendung des Wertes *roundup* kann der Vorlesende entscheiden, ob bei einer Restzahl von Studenten eine Erhöhung oder eine Verringerung der Gruppenstärke für die Aufgaben angemessen ist.

Zunächst überprüft der Algorithmus, ob die Größe der Gruppen angemessen ist. In den anderen Fällen, in denen die Gruppenstärke z. B. die Anzahl der Studenten in der Rangliste übersteigt oder die übergebene Liste leer ist, wird der Algorithmus abgebrochen. Auch wenn die Anzahl überzähliger Studenten bei der Gruppenbildung derart hoch ist, dass nach einem Aufrunden oder Abrunden der Gruppengröße noch überzählige Studenten vorhanden sind, bricht der Algorithmus ab. Anschließend werden einige Hilfsvariablen angelegt. Die Größe der Liste wird benötigt, um die Anzahl der vollen Gruppen und die Restzahl der Studenten zu bestimmen. Zusätzlich wird ein Boolean *reduceSize* benötigt, falls die Gruppengröße reduziert werden soll, um mit den überzähligen Studenten eine Gruppe bilden zu können. Anschließend startet eine Schleife, die läuft, solange noch Gruppen gebildet werden können. Darin wird ein Liste *group* angelegt in die alle Studenten einer Gruppe geschrieben werden sollen. Beim letzten Schleifendurchlauf werden dazu alle verbleibenden Studenten der übergebenen sortierten Liste in *group* geschrieben und die Funktion *matchStudentsInGroup*

aufgerufen. Enthält die Rangliste noch Studenten für mehr als 2 Gruppen werden diese wie folgt gruppiert:

Zunächst wird geprüft ob die Gruppe verkleinert werden soll. Falls das der Fall ist, also *roundup* auf *false* gesetzt ist und es noch überzählige Studenten gibt, soll die nächste gebildete Gruppe einen Studenten weniger als in der Gruppengröße angedacht beinhalten. Dazu wird die *specificGroupSize* anstatt auf die aufgerundete Hälfte der Gruppengröße, auf die aufgerundete Hälfte der um eins verringerten Gruppengröße gesetzt. Dieser Wert wird genommen, um in der folgenden Schleife solange 2 Studenten aus der Rangliste zu entfernen und in *group* zu setzen – jeweils einer aus der leistungsschwachen und einer aus der leistungsstarken Hälfte – bis nur noch ein Student in die Gruppe muss, der aus der Mitte der Rangliste genommen wird, oder die Gruppe voll ist. Abschließend wird geprüft, ob aufgerundet werden soll. Falls es dann noch überzählige Studenten gibt, wird ebenfalls ein Student aus der Mitte der Rangliste genommen und zusätzlich zu *group* hinzugefügt. Danach erfolgt ein Aufruf von *matchStudentsInGroup*. Diese Funktion speichert für jeden Studenten in *group* alle anderen Studenten der Gruppe als Match für die *Lecture* im entsprechenden *StudentPerformanceContainer*.

Bekommen die Algorithmen eine leere Liste oder eine Rangliste mit lediglich einem Studenten übergeben, brechen sie ab und es werden keine Matches gebildet. Zusätzlich muss beim Group-Matching die Gruppengröße für die Anzahl der Studenten angemessen sein. Ebenso sollten alle Algorithmen nur ausgeführt werden, nachdem eine erste *Excercise* durchgeführt und Punkte verteilt wurden. Match-Making mit sämtlichen Algorithmen wären auch möglich, wenn alle Studenten in der Rangliste die gleiche Punktzahl, z. B. 0 Punkte, haben. Allerdings haben die so gebildeten Matches oder Gruppen keinerlei Bezug auf die Leistungsstärke der Studenten, sondern sind einfach zufällig gebildet.

6 Auswertung und Ergebnisse

Die im vorigen Kapitel beschrieben Implementierung der Match-Making-Algorithmen soll im folgenden getestet und die Ergebnisse im Abschnitt 6.1 präsentiert werden. Zusätzlich werden die wichtigsten Punkte der in Kapitel 4 gesammelten Ergebnisse der Befragungen zusammengetragen. Daraus ergibt sich in Abschnitt 6.2 eine Empfehlung für das weitere Verfahren mit dem Programm.

6.1 Evaluation

Um die Algorithmen zu testen, wurden im Programm 50 fiktive Studenten und eine Vorlesung angelegt. Die Studenten besuchen diese Vorlesung. Das bewirkt, dass die Vorlesung eine Liste mit den 50 Studenten beinhaltet. Zusätzlich hat jeder Student einen *StudentPerformanceContainer* für die Vorlesung. Des Weiteren wurde eine *Excercise* erstellt und jedem Studenten für diese Aufgabe eine zufällig erreichte Punktzahl von 0 bis 100 zugeordnet. Danach konnte die Liste der Studenten in der Vorlesung zu einer Rangliste sortiert werden wie in Code 6.1. Bei einer erneuten Ausführung der Sortierung, wird deutlich, dass Studenten mit gleicher Punktzahl zufällig in der Rangliste angeordnet sind, zu sehen in Code 6.2.

Code 6.1: Anfang der Rangliste nach der **ersten** Sortierung

```
1  Name              Points
2  Anton          :  1
3  Nina           :  5
4  Romeo          :  10
5  Markus         :  11
6  Gunther        :  11
7  Tanja          :  12
8  Victoria       :  12
9  Otto           :  12
```

Code 6.2: Anfang der Rangliste nach der **zweiten** Sortierung

```
1  Name              Points
2  Anton          :  1
3  Nina           :  5
4  Romeo          :  10
5  Markus         :  11
6  Gunther        :  11
```

```
  Victoria        :  12
  Otto            :  12
  Tanja           :  12
```

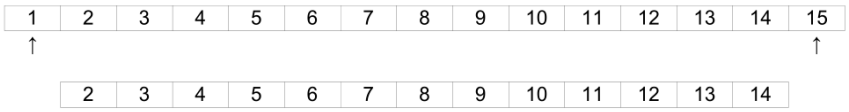


Abbildung 6.1: Die Zuordnung beim control-Matching

Code 6.3: Vergleich von zwei Ausschnitten von zwei Tests des control-Matchings mit derselben aber umsortierten Rangliste

```
----------Test 1 Controlling-Matching------------
Name            :  Points  ; Name            :  Points
Tanja           :  12      ; Gregor          :  74
Victoria        :  12      ; Becky           :  74
Otto            :  12      ; Felix           :  72

----------Test 2 Controlling-Matching------------
Name            :  Points  ; Name            :  Points
Victoria        :  12      ; Becky           :  74
Otto            :  12      ; Gregor          :  74
Tanja           :  12      ; Felix           :  72
```

Der erste Algorithmus ordnet die Studenten paarweise. Dabei wird jeweils mit den beiden Studenten von Anfang und Ende der Rangliste ein Match gebildet, wie in Abbildung 6.1 zu sehen ist. Bei erneuter Sortierung der Liste können sich auf Grund der zufälligen Verteilung der Studenten mit gleicher Punktzahl andere Paare ergeben, wie im Vergleich in Code 6.3 zu sehen ist, indem Otto und Tanja nach Ausführung einer erneuten Sortierung und somit Randomiseierung der Rangliste neue Partner zugewiesen bekommen. Zufälligerweise werden Victoria und Becky in beiden Tests einander zugeordnet, da Victoria zufällig einen Platz in der Rangliste zurückgefallen ist, während Becky aufgerückt ist. Durch die Randomisierung kann sich der Platz in der Rangliste im Verlauf einer *Lecture* unabhängig von der erreichten Punktzahl ändern und somit das Matching von verschiedenen Studenten ermöglichen. Dadurch können Situationen vermieden werden, in denen zwei Studenten immer einander zugeordnet werden. Somit werden eventuelle Defizite beim Erklären oder Verständigungsprobleme, die ein Student mit einem bestimmten anderen Studenten haben kann, ausgeglichen und die Zusammenarbeit mit eventuell Fremden gefördert.

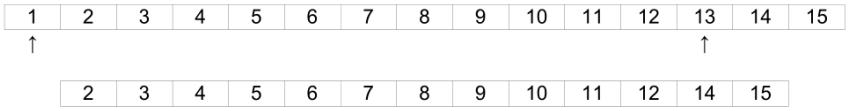


Abbildung 6.2: Die Zuordnung beim immediate-Matching

Code 6.4: Test des immediate-Matching beidem der Student, der zuerst fertig wird, dem schlechtesten zugeordnet wird

```
----------Test Immediate-Matching------------
Name            :  Points  ; Name            :  Points
Anton           :  1       ; Michael         :  82
Nina            :  5       ; Willi           :  94
Romeo           :  10      ; Lucy            :  69
Markus          :  11      ; Felix           :  72
Gunther         :  11      ; Susi            :  95
```

Der zweite Algorithmus zielt darauf ab, einen Studenten, der eine Aufgabe schon fertig bearbeitet hat, dem schlechtesten Studenten zuzuordnen, da dieser am wahrscheinlichsten Hilfe benötigt. Wie in Abbildung 6.2 und Code 6.4 zu sehen, ist nicht der beste Student zuerst fertig, sondern ein anderer. Dieser mit der Nummer 13 in der Abbildung 6.2 wird dem schlechtesten mit der Nummer 1 zugeordnet. Somit können schon während der Bearbeitung der Aufgabe die hilfebedürftigen Studenten unterstützt werden. Die zufällige Verteilung der Studenten in der Rangliste spielt bei diesem Algorithmus eine untergeordnete Rolle, da niemand vorhersagen kann, wer eine Aufgabe am schnellsten bearbeitet. Er erfüllt aber den selben Zweck wie beim *control-Matching*, wenn die Abfolge der Studenten, die fertig werden, sich exakt mit dem oberen Ende der Rangliste deckt.

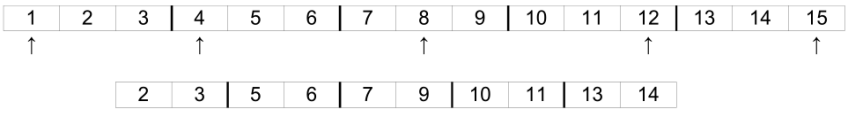


Abbildung 6.3: Die Zuordnung beim group-Matching

Code 6.5: 4 durch Match-Making gebildete Gruppen bei einer Gruppenstärke von 4 und dem Aufrunden von Gruppen

```
----------Test Group-Matching------------
------Group 12------
Anton           :  1
Susi            :  95
Xena            :  19
Zelda           :  66
Erik            :  36
------Group 11------
Nina            :  5
Willi           :  94
Tristan         :  22
Elisabeth       :  65
Pia             :  38
------Group 10------
Romeo           :  10
Karl            :  93
Yvonne          :  23
Svenja          :  62
```

```
------Group 9------
Markus          :  11
Julia           :  86
Ronny           :  23
Nancy           :  60
```

Der dritte Algorithmus ordnet die Student in beliebig große Gruppen. Die Rangliste wird dafür in der Gruppengröße entsprechend in Teillisten geteilt, was durch die dicken Linien in Abbildung 6.3 dargestellt ist. Somit kann aus jeder dieser Teillisten ein Student gewählt und aus diesen eine Gruppe gebildet werde. Dadurch ist gewährleistet, dass eine Gruppe immer aus leistungsstärkeren und leistungsschwächeren Studenten gleichermaßen besteht. Die gesamte Leistungsstärke orientiert sich dabei am Median der Studenten. Zusätzlich kann gewählt werden, ob bei einer Restzahl von Studenten einige Gruppen vergrößert oder verkleinert werden sollen. Code 6.5 zeigt das Vergrößern einiger Gruppen bei einer eigentlichen gewählten Gruppenstärke von 4. Auch hier bringt die zufällige Sortierung von Studenten einen Vorteil. Sie sorgt dafür, dass die Studenten zumeist unterschiedlichen Gruppen zugeordnet werden, da nur in sehr wenigen Fällen – z. B. in der zuletzt gebildeten Gruppe – Studenten die in aufeinanderfolgenden Plätzen in der Rangliste stehen, in dieselbe Gruppe kommen. Somit können sich mit dem Platz in der Rangliste auch die Studenten, mit denen zusammengearbeitet werden soll, ändern.

In den zwei Befragungen sind die Studenten eher wenig auf das Match-Making an sich eingegangen. Sie scheinen dem gegenüber nicht abgeneigt zu sein und wünschen sich durchaus ein Match-Making – hauptsächlich für die Gruppenbildung – und könnten sich auch vorstellen Lerngruppen so bilden zu lassen. Die stetige Rückmeldung über den eigenen Wissenstand, der als Grundlage für das Match-Making erfasst werden muss, sehen sie als positiv. Somit kann die Motivation der Studenten für einzelne Fächer mehr zu lernen erhöht werden und die gleichzeitige Hilfe von leistungsstarken Studenten kann zu besserem Verständnis für den Lehrstoff führen und zusätzlich motivieren. Das wichtigste für die Studenten und die Motivation ein solches Programm zu verwenden ist jedoch die Nutzbarkeit. Sie würden nur ungern zu viel Freizeit für ein solches Programm opfern. Auch wünschen sie sich ein ansprechendes User-Interface und weitere nützliche Funktionen, die in das Programm integriert werden, wie z. B. ein Stundenplan, Materialien zu den Vorlesungen oder Seminaren sowie Neuigkeiten im jeweiligen Fachbereich für die Interessierten, einen Chatroom für die Vorlesung um direkt Fragen an Kommilitonen richten zu können und noch vieles Anderes. Das zeigt auch ein anderer Wunsch der Studenten, der am ehesten über die Verwendung des Programms bei Studenten entscheiden wird: sie wünschen sich eine Vereinheitlichung der vielen Programme und Websites, die es schon gibt und die Informationen, Materialien und anderes bieten, zusätzliche praktische Aufgaben bereitstellen oder für Einschreibungen genutzt werden. Einen zentralen Punkt mit eventuellen Links zu entsprechenden Websites und Programmen fänden sie ansprechend.

6.2 Empfehlung

Die vorgestellten und programmierten Match-Making-Algorithmen können Studentengruppen in Paare oder Gruppen einteilen. Je nach Art der Lehrveranstaltung und der zu bearbeitenden Aufgaben kann somit gewählt werden, welcher Algorithmus eingesetzt wird. So bietet es sich z. B. für Vorlesungen mit kleineren Programmieraufgaben an, einzelne Studenten mit dem *immediate Matching* sofort nach Fertigstellung ihrer Aufgabe den Studenten zuzuordnen, die am ehesten Hilfe benötigen. Die hilfsbedürftigen Studenten sind die am unteren Ende der Rangliste. Mit Hilfe der durch Aufgaben gewonnenen Punkte lässt sich diese Rangliste immer weiter präzisieren. Somit wäre es von Vorteil, bevor das *immediate Matching* genutzt wird, die Studenten zunächst Punkte sammeln zu lassen, indem sie Aufgaben wie Multiple-Choice-Tests bearbeiten. Diese können zusätzlich mit Hilfe des *control Matchings* von Studenten gegenseitig kontrolliert werden. Ein solches Match-Making ist sehr einfach einsetzbar, für viele Aufgaben und in fast allen möglichen Arten von Lehrveranstaltungen. Auch das *Group-Matching* sollte auf einer Rangliste durhgeführt werden, die mindestens eine grobe Einteilung der Studenten darstellt. Je feiner unterteilt die Rangliste ist, umso gleichmäßiger werden die Studenten den Gruppen zugewiesen. Somit sind in jeder Gruppe gleich viele leistungsstarke und -schwache Studenten. Wenn es gewünscht wird können auch Lerngruppen mit Hilfe dieses Algorithmus gebildet werden.

Also sollte vor dem ersten Match-Making in jeder Lehrveranstaltung schon eine Aufgabe bearbeitet und bewertet worden sein, damit die Algorithmen nicht auf Ranglisten ausgeführt werden, in denen alle Studenten den gleichen Score haben. Die verschiedenen möglichen Arten der Aufgaben wurden in dieser Arbeit lediglich grob umrissen und auf anwendbare Match-Making-Algorithmen untersucht. Bei der Erstellung von Aufgaben sollte darauf geachtet werden, welche Möglichkeiten sich daraus für ein Match-Making ergeben oder ob in Tests andere Algorithmen dafür benötigt werden. Diese können anschließend der Klasse *Matching* hinzugefügt werden.

Auf Grund der Studentenbefragungen ist vor allem auf die Entwicklung eines guten User-Interfaces zu achten und weitere Elemente von Gamification, neben den Punkten, einzuführen. Das können Level sein, die dargestellt werden oder Zielwerte, die die Studenten sich selbst setzen können und Abzeichen für gute Erklärungen oder Hilfen, welche später als zusätzliche Dimension im Match-Making genutzt werden könnten. Solche Elemente fördern die Motivation der Studenten das Programm zu nutzen und somit auch die Effektivität des Match-Makings und unterstützen letztlich den Wissenserwerb der Studenten. Auch sollte das Programm leicht erweiterbar sein, um Features, welche von den Studenten gewünscht werden, einbauen zu können. Daher sollte nach einer Testphase eine erneute Befragung der Studenten, die das Programm genutzt haben, durchgeführt werden. Letztlich wünschten sich die Studenten auch eine Vereinheitlichung der Strukturen und eine niedrigere Zahl an zusätzlichen Angeboten für das universitäre Lernen. Es sollte überprüft werden, welche Websites oder Programme genutzt werden und ob sich ein oder mehrere davon zusammenfassen lassen. Dadurch würden die Studenten nicht von einem Überangebot abgeschreckt.

7 Zusammenfassung und Ausblick

Der Wissensstand der Studenten wird an Universitäten zumeist erst am Ende eines Semesters in Prüfungen getestet. Rückmeldung zu ihrem Leistungsstand in verschiedenen Lehrgebieten erhalten die Studenten somit erst dann, wenn die zugehörige Vorlesung in dem Semester schon vorbei ist. Üblicherweise wird dieselbe Vorlesung erst im übernächsten Semester erneut angeboten. Mit Hilfe von kleinen Tests und Programmieraufgaben kann Studenten schon im Verlauf eines Semester in einer Vorlesung oder einem Seminar eine Rückmeldung über ihren Leistungsstand gegeben werden. Jedoch erfordert die Kontrolle der Aufgaben und die Hilfe für Studenten, welche darin schlecht abgeschnitten haben, einiges an Aufwand und Zeit von den Mitarbeitern. Die Hilfe für leistungsschwache Studenten und eine teilweise Kontrolle der Aufgaben kann jedoch auch von den Studenten selbst vorgenommen werden. Dazu wird ein Match-Making benötigt, welches leistungsstarke Studenten zu leistungsschwachen zuordnet, damit sie die kontrollieren und ihnen gegebenenfalls helfen können, indem sie den Lösungsweg erklären und das dazu nötige Wissen vermitteln. In dieser Arbeit wurden verschiedene Protokolle und Algorithmen zum Match-Making betrachtet und das Match-Making in Computerspielen als sinnvoll für Peer Assessment im universitären Kontext erachtet, da es auf einer leicht aufzubauenden Rangliste basiert und nicht zu komplex ist oder zu viel Anlaufzeit erfordert. Eine solche Rangliste kann in Vorlesungen und Praktika gleichermaßen erstellt werden, indem kleine Tests, z. B. Multiple Choice Tests, oder praktische Aufgaben und Teilaufgaben gestellt und bewertet werden. Daraufhin können verschiedene Arten des Match-Makings auf der Rangliste ausgeführt werden. Drei Algorithmen wurden dafür auf der Grundlage des Match-Makings in Computerspielen konzipiert und prototypisch implementiert: das kontrollierende Matching, das sofortige Match-Matching und Gruppen-Matching. Zusätzlich wurden zwei Befragungen von Studenten durchgeführt. Die erste war allgemein und wurde vor der Konzeption und Implementierung durchgeführt. Es hat sich dabei herausgestellt, dass die Studenten ein Programm mit Match-Making nutzen würden und gerne eine Rückmeldung über ihrem Lernfortschritt hätten. Die Studenten in der zweiten Befragung sollten selbst Arten für das Match-Making auf einer Rangliste erarbeiten oder die Anforderungen der Studenten an ein solches Programm diskutieren. Die Ergebnisse konnten zum Teil schon in die Implementierung einfließen, zum großen Teil jedoch bezogen sich die Wünsche der Studenten auf die Gestaltung des User Interfaces und mögliche Features anstatt auf das Match-Making.

In weiteren Arbeiten sollte daher der Fokus auf der Fertigstellung des Programms liegen. Dazu wird einerseits eine Vielzahl von möglichen Aufgaben benötigt, die in Vorlesungen gestellt werden können und deren Bewertung einfach ist und somit automatisiert oder von Studenten ausgeführt werden kann. Andererseits benötigt ein solches Programm ein sehr

gutes User Interface um sich mit schon vorhandenen Angeboten messen zu können und von Studenten aktiv genutzt zu werden. Auch sollte mögliches Match-Making und Punkterwerb in Übungen diskutiert werden, welche auf Grund der schon vorhanden Interaktivität und der schon gestellten Aufgaben Möglichkeiten dafür bieten, als in Vorlesungen oder Seminaren und Praktika.

Die Aufgaben sollten ansprechend und hilfreich gestaltet werden. Dazu könnten zusätzlich verschiedene Schwierigkeitsgrade und Hilfestellung wie Tipps angeboten werden. Weiterhin könnten zeitintensive Aufgaben gestellt und mit Meilensteinen versehen werden, um den Fortschritt festzuhalten und durch das sofortige Matching fertige Studenten immer den langsamsten zuzuordnen. Ebenso würde es die Motivation und das Wissen der Studenten steigern, Hilfsanfragen und -angebote einstellen zu können und danach gematcht zu werden. Die Meilensteine könnten als Abzeichen, oder auch Badges genannt, im Kontext von Gamification je nach Wünschen der Studenten in das Programm integriert werden. Solche Abzeichen können für gute Erklärungen oder Hilfen von den leistungsschwachen Studenten vergeben werden und eventuell als zusätzliche Dimension für das Match-Making genutzt werden, damit der schwächste Student dem besten Studenten zugeordnet wird, der gleichzeitig viele Abzeichen für gutes Erklären hat.

Des Weiteren sollten die in den Befragung geäußerten Wünsche der Studenten umgesetzt werden, soweit sie das User Interface und Gamification betreffen. Zusätzlich gewünschte Features sollten erst nach einem Testlauf implementiert werden. Elemente von Gamification motivieren die Studenten zur Verwendung des Programms und können somit dazu dienen die Leistungen der Studenten das Semester über zu verbessern und nachzuvollziehen.

Nach der Fertigstellung des Programms sollten erneut Studentenbefragungen erfolgen, die Nutzbarkeit und Häufigkeit der Anwendung messen, sowie das Programm hinsichtlich Wissensvermittlung und Motivation der Studenten testen. Auch die Lernkurven der Studenten und ihre Fähigkeiten zum Teamwork können überprüft werden. Abschließend kann die Befragung Möglichkeiten für zusätzliche Motivation der Studenten hervorbringen. Eine solche Motivation wäre, dass Zusatzpunkte für Prüfungen gesammelt werden könnten.

Literatur

[TW12] A.S. Tanenbaum und D.J. Wetherall. „Computernetzwerke“. In: Always learning. Pearson, 2012, S. 846–856. ISBN: 9783868941371.

[YSH03] Fan Yang, Rui-min Shen und Peng Han. „A novel self-organizing E-Learner community model with award and exchange mechanisms“. In: Feb. 2003.

[Yan+06] Fan Yang u. a. „An Improved E-learner Community Construction Algorithm Based on Learning Interest Feature Vectors“. In: *Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications*. AIC'06. Elounda, Greece: World Scientific, Engineering Academy und Society (WSEAS), 2006, S. 254–259. ISBN: 960-8457-51-3. URL: http://dl.acm.org/citation.cfm?id=1366421.1366465.

[Cre+05] Raquel M. Crespo u. a. „An Algorithm for Peer Review Matching Using Student Profiles Based on Fuzzy Classification and Genetic Algorithms“. In: Hrsg. von Moonis Ali und Floriana Esposito. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 685–694. ISBN: 978-3-540-31893-4. DOI: 10.1007/11504894_95. URL: http://dx.doi.org/10.1007/11504894_95.

[Gam16] Riot Games. „Leitfaden zum Matchmaking“. In: 21. Juli 2016. URL: https://support.riotgames.com/hc/de/articles/201752954-Leitfaden-zum-Matchmaking.

[VMM14] Maxime Véron, Olivier Marin und Sébastien Monnet. „Matchmaking in Multiplayer On-line Games: Studying User Traces to Improve the User Experience“. In: *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. NOSSDAV '14. Singapore, Singapore: ACM, 2014, 7:7–7:12. ISBN: 978-1-4503-2706-0. DOI: 10.1145/2578260.2578265. URL: http://doi.acm.org/10.1145/2578260.2578265.

[Bei14] Lars Beier. „Evaluating the Use of Gamification in Higher Education to Improve Students Engagement“. In: März 2014.

[Mor] Martin Morgenstern. „Praktomat“. In: URL: https://praktomat.inf.tu-dresden.de/.

[Krü13] Marc Krüger. „Erstellen und Bewerten von Multiple-Choice-Aufgaben“. In: 2013. URL: https://www.uni-hannover.de/fileadmin/luh/content/elearning/practicalguides2/didaktik/elsa_handreichung_zum_erstellen_und_bewerten_von_mc-fragen_2013.pdf.

[Bro86] John Brooke. „SUS - A quick and dirty usability scale“. In: Redhatch Consulting Ltd., 1986.

1. Das zu entwickelnde Programm, soll in Vorlesungen, Seminare und Praktika Verwendung finden.

Ziel ist, den Lernfortschritt durch Punkte (eventuell Level) und Abzeichen sichtbar zu machen. Punkte sollen durch verschiedene kleinere Aufgaben (z.B. Multiple Choice Test, kurze Programmieraufgaben, ...) erworben werden können. Abzeichen sollen durch Fortschritte bei komplexeren Aufgaben, sowie Interaktion mit anderen erworben werden können. Hat ein Student Probleme mit einer Aufgabe, kann ihm ein anderer, der die Aufgabe schon bewältigt hat, helfen. Diese Zuordnung erfolgt durch Matching.

Bitte geben Sie an, in wie weit Sie den Aussagen zustimmen.

	Stimme nicht zu				Stimme voll und ganz zu	Kann ich nicht beurteilen
1. Ich kann mir sehr gut vorstellen, das Programm regelmäßig zu nutzen.	○	○	○	○	○	○
2. Ich empfindefinde das Programm-Konzept als unnötig komplex.	○	○	○	○	○	○
3. Ich empfinde das Programm-Konzept als nützlich.	○	○	○	○	○	○
4. Es macht mir nichts aus bewertet zu werden.	○	○	○	○	○	○
5. Ich möchte nicht gematcht werden.	○	○	○	○	○	○
6. Ich finde es gut, dass verschiedene Funktionen (Punkte, Abzeichen, Matching,...) in einem Programm integriert sind.	○	○	○	○	○	○
7. Ich habe mir so ein Programm schon gewünscht.	○	○	○	○	○	○
8. Ich würde eher offline/handschriftlich arbeiten wollen.	○	○	○	○	○	○
9. Ich finde es gut, zeitnah automatisierte Rückmeldungen zu meinem Lernbedarf/Lernfortschritt zu erhalten.	○	○	○	○	○	○
10. Mir reichen die bisherigen Angebote (Übungen, Auditorium) aus.	○	○	○	○	○	○

2. Ich wünsche mir in einem solchen Programm:

- ☐ Punkte
- ☐ Level
- ☐ Abzeichen
- ☐ Lerngruppen-Matchmaking
- ☐ Anderes, und zwar

3. Geben sie bitte Ihr Alter an

Alter: Jahre

4. Bitte geben Sie Ihren Studiengang an.

5. In welchem Fachsemester studieren Sie?

Fachsemester

Vielen Dank für Ihre Teilnahme =)

Wenn Sie noch Anmerkungen haben, können Sie diese gerne notieren.

Anmerkung:

Häufigkeiten

[DatenSet1] C:\Users\Paddy\Desktop\Druck\Kurzbefragung\Auswertung KB 1\Kurzbefragung 1 Auswertung.sav

Statistiken

		Ich kann mir sehr gut vorstellen, das Programm regelmäßig zu nutzen.	Ich empfinde as Programmkonzept NICHT als unnötig komplex.	Ich empfinde das Programm-Konzept als nützlich	Es macht mir nichts aus bewertet zu werden.	Ich MÖCHTE gematcht werden.
N	Gültig	68	70	71	73	70
	Fehlend	10	8	7	5	8
Mittelwert		3,2647	3,7143	3,6056	3,4384	3,1714
Median		4,0000	4,0000	4,0000	4,0000	3,0000
Modus		4,00	4,00	4,00	4,00	3,00

Statistiken

		Ich finde es gut, dass verschiedene Funktionen in einem Programm integriert sind.	Ich habe mir so ein Programm schon gewünscht.	Ich würde eher NICHT offline/handschriftlich arbeiten wollen.	Ich finde es gut, zeitnah automatisierte Rückmeldungen zum Lernvortschritt zu erhalten	Mir reichen die bisherigen Angebote NICHT.
N	Gültig	70	70	74	75	75
	Fehlend	8	8	4	3	3
Mittelwert		3,5714	2,2000	3,2703	3,8400	3,0667
Median		4,0000	2,0000	3,0000	4,0000	3,0000
Modus		4,00	1,00	3,00	4,00	3,00

Häufigkeitstabelle

Häufigkeiten

[DatenSet1] C:\Users\Paddy\Desktop\Druck\Kurzbefragung\Auswertung KB 1\Kurzbefragung 1 Auswertung.sav

Statistiken

		K1_score	K1_score_niv
N	Gültig	49	74
	Fehlend	29	4
Mittelwert		34,2245	32,8919
Median		36,0000	34,0000
Modus		36,00	29,00

Häufigkeitstabelle

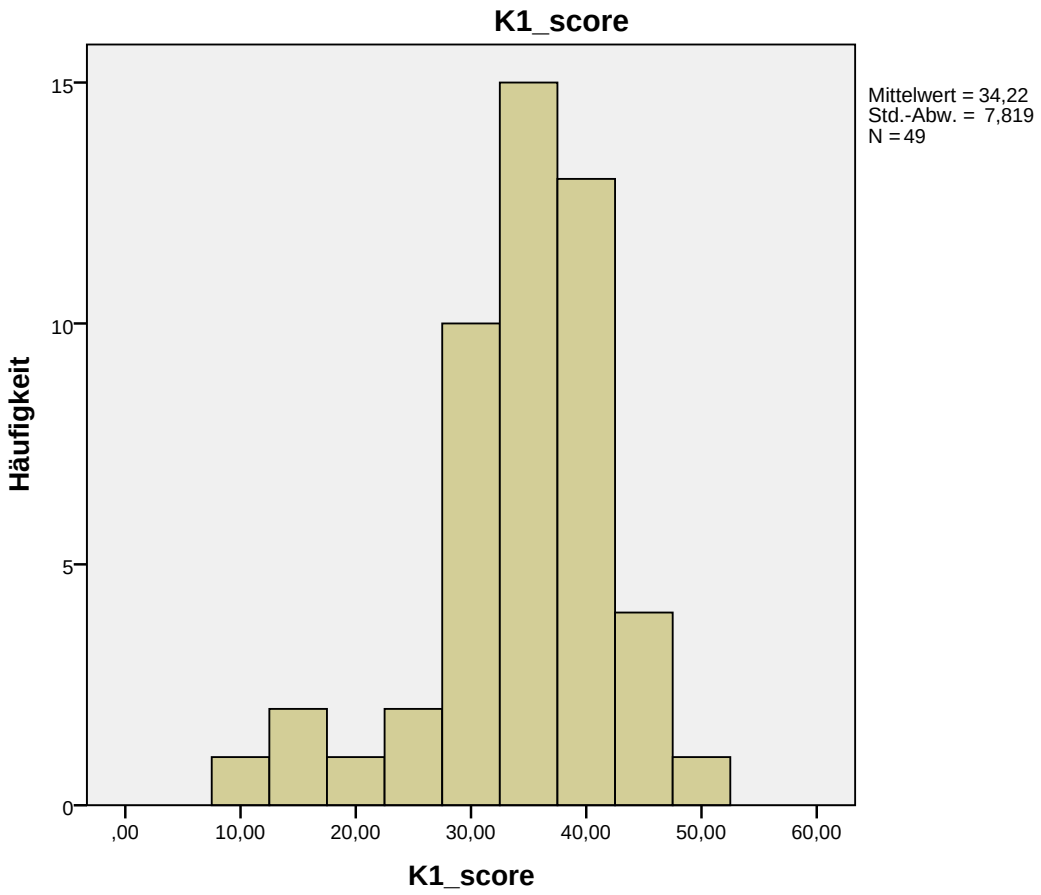
K1_score

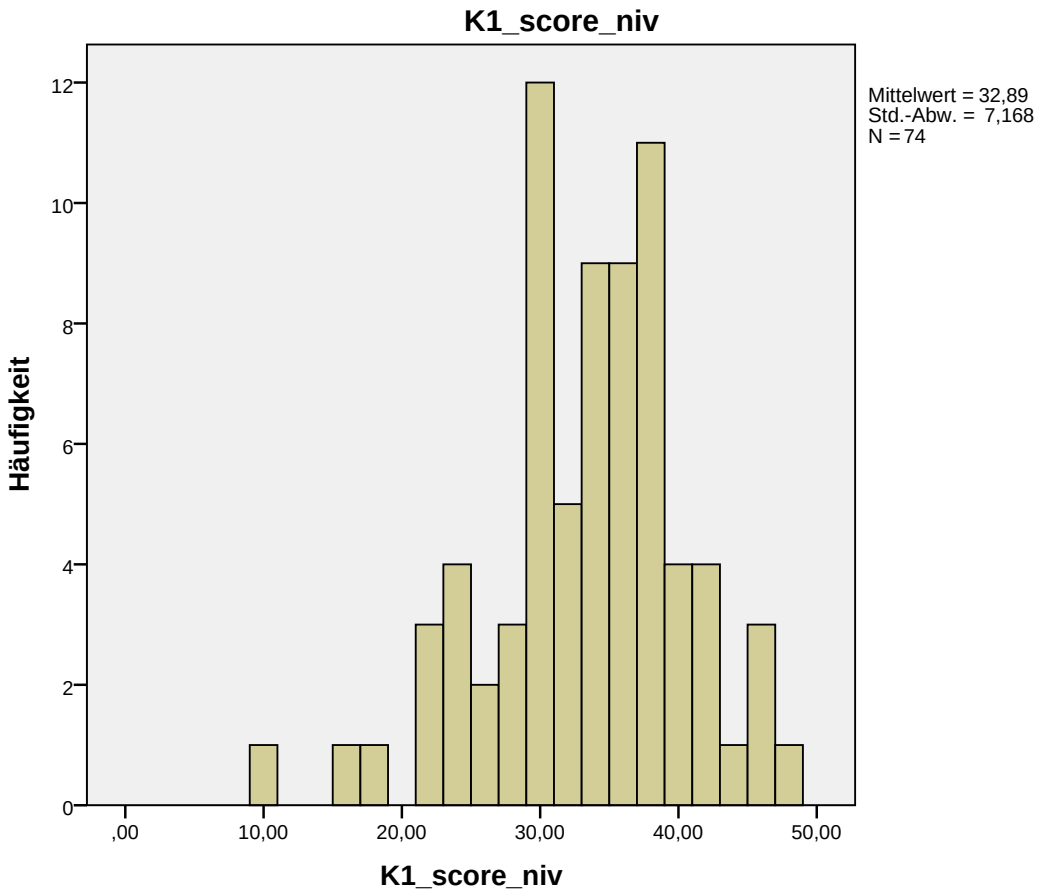
		Häufigkeit	Prozent	Gültige Prozente	Kumulierte Prozente
Gültig	10,00	1	1,3	2,0	2,0
	16,00	1	1,3	2,0	4,1
	17,00	1	1,3	2,0	6,1
	21,00	1	1,3	2,0	8,2
	23,00	1	1,3	2,0	10,2
	26,00	1	1,3	2,0	12,2
	28,00	2	2,6	4,1	16,3
	29,00	5	6,4	10,2	26,5
	30,00	2	2,6	4,1	30,6
	31,00	1	1,3	2,0	32,7
	33,00	1	1,3	2,0	34,7
	34,00	3	3,8	6,1	40,8
	35,00	2	2,6	4,1	44,9
	36,00	7	9,0	14,3	59,2
	37,00	2	2,6	4,1	63,3
	38,00	5	6,4	10,2	73,5
	39,00	3	3,8	6,1	79,6
	40,00	1	1,3	2,0	81,6
	41,00	3	3,8	6,1	87,8
	42,00	1	1,3	2,0	89,8
	44,00	1	1,3	2,0	91,8
	45,00	1	1,3	2,0	93,9
	46,00	2	2,6	4,1	98,0
	48,00	1	1,3	2,0	100,0
	Gesamt	49	62,8	100,0	
Fehlend	System	29	37,2		
Gesamt		78	100,0		

K1_score_niv

		Häufigkeit	Prozent	Gültige Prozente	Kumulierte Prozente
Gültig	10,00	1	1,3	1,4	1,4
	16,00	1	1,3	1,4	2,7
	17,00	1	1,3	1,4	4,1
	21,00	2	2,6	2,7	6,8
	22,00	1	1,3	1,4	8,1
	23,00	1	1,3	1,4	9,5
	24,00	3	3,8	4,1	13,5
	26,00	2	2,6	2,7	16,2
	28,00	3	3,8	4,1	20,3
	29,00	9	11,5	12,2	32,4
	30,00	3	3,8	4,1	36,5
	31,00	4	5,1	5,4	41,9
	32,00	1	1,3	1,4	43,2
	33,00	3	3,8	4,1	47,3
	34,00	6	7,7	8,1	55,4
	35,00	2	2,6	2,7	58,1
	36,00	7	9,0	9,5	67,6
	37,00	5	6,4	6,8	74,3
	38,00	6	7,7	8,1	82,4
	39,00	3	3,8	4,1	86,5
	40,00	1	1,3	1,4	87,8
	41,00	3	3,8	4,1	91,9
	42,00	1	1,3	1,4	93,2
	44,00	1	1,3	1,4	94,6
	45,00	1	1,3	1,4	95,9
	46,00	2	2,6	2,7	98,6
	48,00	1	1,3	1,4	100,0
	Gesamt	74	94,9	100,0	
Fehlend	System	4	5,1		
Gesamt		78	100,0		

Histogramm


K1_score
Mittelwert = 34,22
Std.-Abw. = 7,819
N = 49
Häufigkeit
15
10
5
0
,00
10,00
20,00
30,00
40,00
50,00
60,00
K1_score


K1_score_niv
Häufigkeit
0
2
4
6
8
10
12
,00
10,00
20,00
30,00
40,00
50,00
K1_score_niv
Mittelwert = 32,89
Std.-Abw. = 7,168
N = 74

Mehrfachantworten

[DatenSet1] C:\Users\Paddy\Desktop\Druck\Kurzbefragung\Auswertung KB 1\Kurzbefragung 1 Auswertung.sav

Fallzusammenfassung

	Fälle					
	Gültig		Fehlend		Gesamt	
	N	Prozent	N	Prozent	N	Prozent
$K2_set[a]	66	84,6%	12	15,4%	78	100,0%

a. Dichotomie-Gruppe tabellarisch dargestellt bei Wert 1.

Häufigkeiten von $K2_set

		Antworten		Prozent der Fälle
		N	Prozent	
$K2_set[a]	Punkte	48	35,3%	72,7%
	Level	37	27,2%	56,1%
	Abzeichen	19	14,0%	28,8%
	Lerngruppen-Matching	24	17,6%	36,4%
	Anderes, und zwar:	8	5,9%	12,1%
Gesamt		136	100,0%	206,1%

a. Dichotomie-Gruppe tabellarisch dargestellt bei Wert 1.

Codebuch

[DatenSet1] C:\Users\Paddy\Desktop\Druck\Kurzbefragung\Auswertung KB 1\Kurzbefragung 1 Auswertung.sav

K2_5

		Wert	Anzahl	Prozent
Standardattribute	Position	15		
	Label	Anderes, und zwar:		
	Typ	Numerisch		
	Format	F8.2		
	Messung	Nominal		
	Rolle	Eingabe		
Gültige Werte	,00	Nicht gewählt	67	85,9%
	1,00	gewählt	8	10,3%
Fehlende Werte	99,00	Keine Angabe	3	3,8%

K2_5a

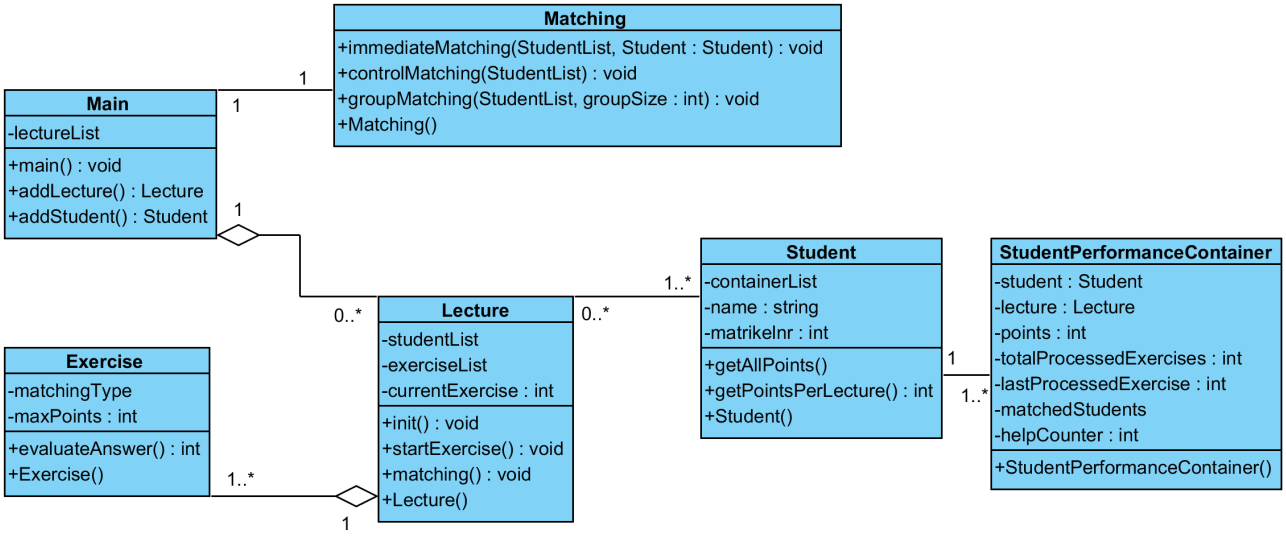
		Wert	Anzahl	Prozent
Standardattribute	Position	16		
	Label	Anderes (offene Eingabe)		
	Typ	String		
	Format	A750		
	Messung	Nominal		
	Rolle	Eingabe		
Gültige Werte			72	92,3%
	das ersetzen anderer Lernplattformen und übernahme derer features. Zu viele Mögliche Plattformen existieren. Eine große wäre sinnvoller		1	1,3%
	Looten		1	1,3%
	Musterlösung für Aufgaben und wenn nötig Hinweise zum Lösen		1	1,3%
	nix		1	1,3%
	Übungsaufgaben mit Kontrolle		1	1,3%
	vielleicht auch Klausur-Bonuspunkte		1	1,3%

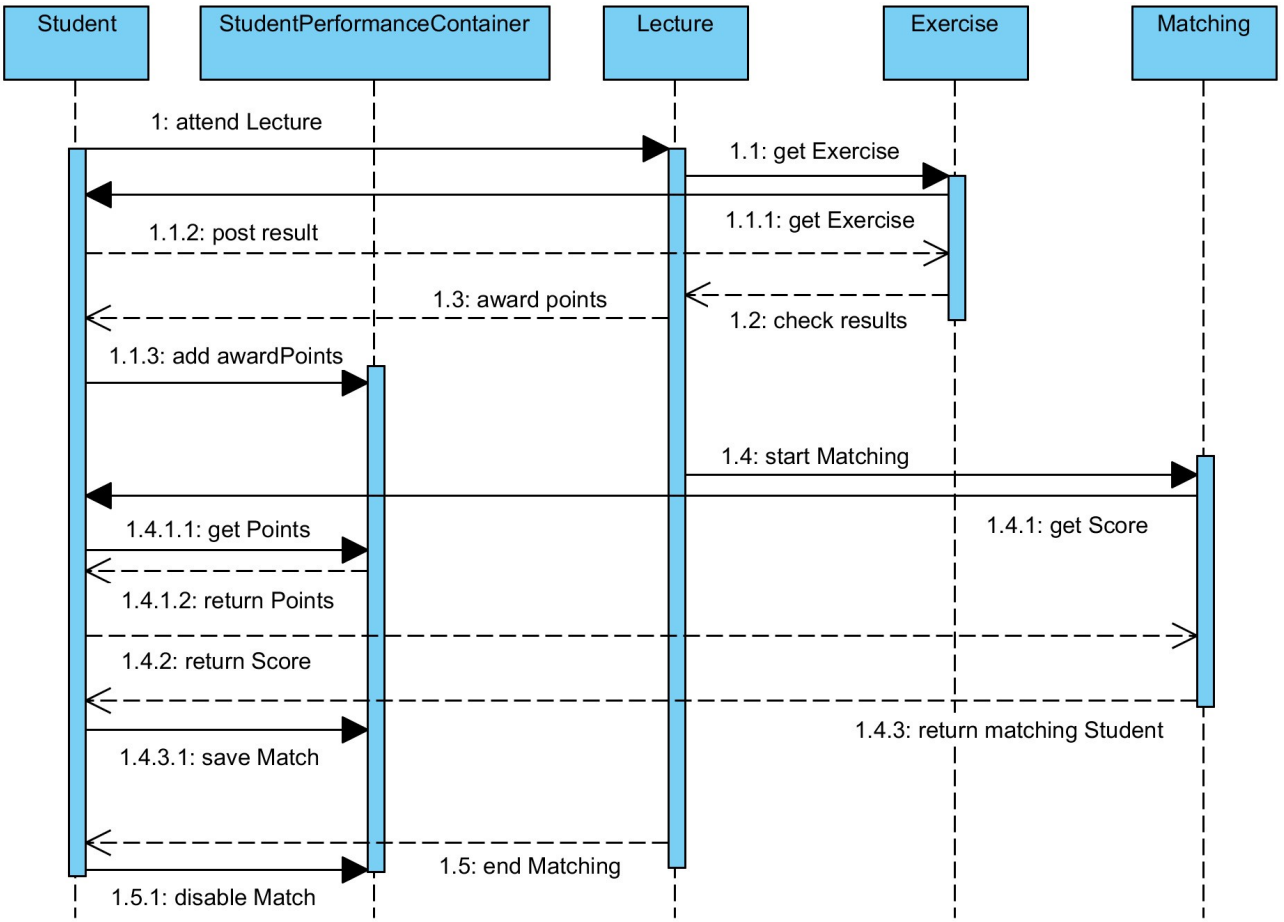
SD_end

		Wert	Anzahl	Prozent
Standardattribute	Position	20		
	Label	Schlusfrage (offene Eingabe)		
	Typ	String		
	Format	A750		
	Messung	Nominal		
	Rolle	Eingabe		
Gültige Werte			66	84,6%
	[SD_1 "n; n>0"]		1	1,3%
	DA WAR EIN RECHTSCHREIBFEHLER; FIXED THAT FOR ...		1	1,3%
	Der Inhalt des Prgs fehlt. Was soll gelernt werden?		1	1,3%
	Frage 2 hat einen Fehler: empfindefinde Woher soll ich wissen, was ich mir wünsche, wenn ich vor 2 Minuten noch nichts von dieser Idee wusste? Muss echt alles von mir bewertet werden? Was ist, wenn ich anderen gar nicht helfen will?		1	1,3%
	Halte so ein Programm für eher unnötig, aber viel Spaß!		1	1,3%
	https		1	1,3%
	Ich möchte nichts anmerk		1	1,3%
	Info-Studium schon einfach genug!		1	1,3%
	möglichst geringer zusätzlicher Zeitaufwand		1	1,3%
	Sollen die Punkte für die Klausur gelten? Davon mache ich alle Antworten abhängig.		1	1,3%
	Sowas wie Praktomat für andere Fächer wäre gut.		1	1,3%
	Super Idee!!! [K1_8 "aber Datenschutz muss gewahrt sein"; K1_10 Auditorium" Wer nutzt das schon?"]		1	1,3%

Group 1: Ideas of Matching

The program, which has to be developed, should be used in lectures, seminars and practica. The goal is to visualize the learning progress via points, maybe levels and badges. Points should be earned through the solving of minor tasks like short multiple choice tests, programming tasks, etc. Badges should be earned through progress in more complex tasks, as well as interaction with other students. For example: if a student encounters problems while solving a task, another student, who has already finished that task, can help. The necessary mapping of these student will be done through matching.



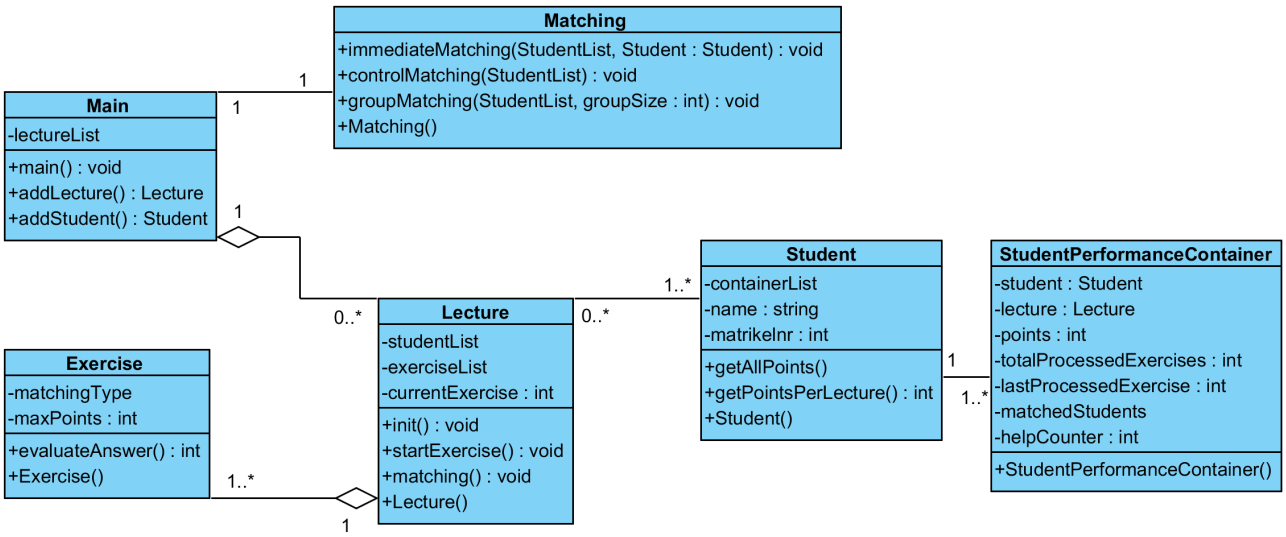


Please discuss the Matching in this program.

1. What would you expect from a matching system, that should match students, so they can help each other?
2. Are there different styles of Matching that should be used? Describe them.

Group 2: UML-Diagram

The program, which has to be developed, should be used in lectures, seminars and practica. The goal is to visualize the learning progress via points, maybe levels and badges. Points should be earned through the solving of minor tasks like short multiple choice tests, programming tasks, etc. Badges should be earned through progress in more complex tasks, as well as interaction with other students. For example: if a student encounters problems while solving a task, another student, who has already finished that task, can help. The necessary mapping of these student will be done through matching.



This picture shows the classes for the program and their relations. The Classes are described in the following:

Student: This class contains information about the student as the name, student number, term, etc. A Student can attend multiple lectures in a term. All data regarding a lecture are saved in an extra container. The student possesses a list of all these containers.

Lecture: This class should represent a lecture, seminar or practicum. Such a course is valid for one term and can be a following course or be a basic course or part of a series of courses, e.g. Math 1, Math 2 and Math 3. Therefore Points from basic courses should be partially carried over and serve as a first order of the students for the matching.

Exercise: There are diverse forms of tasks or exercises, which can be used for e.g. Multiple Choice Tests, programming tasks, etc. But there should be always a matchingtype which says, which algorithm of matching is used for this exercise. Also the maximum of points, which can be earned by fulfilling the task correctly, has to be defined. The answers students give have to be checked, but that can be handled differently for each form of task.

StudentPerformanceContainer: This container is created for every course a student attends and should save all data a student has for a course. It contains the points the student has earned and the number of tasks performed. Additionally there is counter which shows how often the student has helped others. If there is an active matching which assigns the student to one or more other students they will be saved in a list.

Matching: This class provides different matching algorithms because exercises can have various requirements for a matching. The appropriate algorithm should be chosen in the *exercise* class or the *lecture* class, if a matching should be done without a specific task.

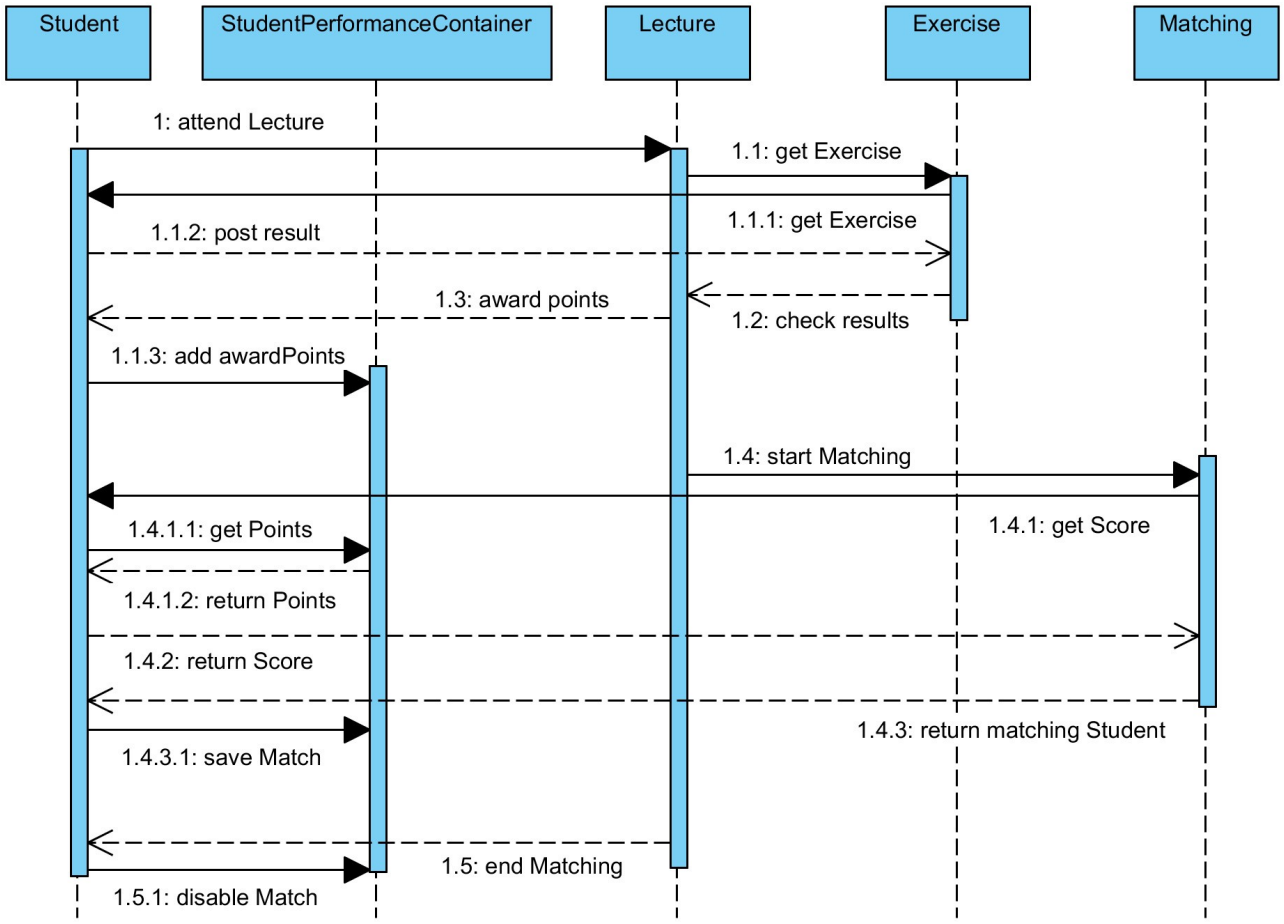
Please discuss the UML-Diagram.

1. Does the UML-Diagram fulfill the requirements for the program? Explain!
2. Is it extensible enough for different future tasks? Discuss useful extensions!

Group 3: The Students

Imagine you are a student, who should use the following program:

The program, which has to be developed, should be used in lectures, seminars and practica. The goal is to visualize the learning progress via points, maybe levels and badges. Points should be earned through the solving of minor tasks like short multiple choice tests, programming tasks, etc. Badges should be earned through progress in more complex tasks, as well as interaction with other students. For example: if a student encounters problems while solving a task, another student, who has already finished that task, can help. The necessary mapping of these student will be done through matching.



In this diagram you can see the interaction with the program. When *students* attend a *lecture*, they get exercises, they have to do. The results are checked and for correct answers points are awarded. The points are added to the points a *student* already has earned in this *lecture* and saved in the *StudentPerformanceContainer*.

Immediately or later, depending on the *exercise*, the *lecture* starts a *matching*. The algorithm used for the matching is chosen by the kind of the *exercise* or the *lecture* itself when there is no related *exercise*. For the computation of the matching, all the points from the *students* that they have earned for this *lecture*, are needed. This so called score is collected from the *StudentPerformanceContainer*. In the end, the match which can be one or more *students*, is given to the *student* and also stored in the *StudentPerformanceContainer*. The matching is active until the *lecture* ends it.

Imagine you are a student using a program like this.

1. What would you, as the student, like to see? For example: Your points? The level for each *lecture*? Discuss!
2. What will make you use a program like this more often?

```
Name              Points
Anton          :  1
Nina           :  5
Romeo          :  10
Markus         :  11
Gunther        :  11
Tanja          :  12
Victoria       :  12
Otto           :  12
Zoey           :  15
Maria          :  15
Quentin        :  16
Moritz         :  19
Xena           :  19
Tristan        :  22
Yvonne         :  23
Ronny          :  23
Lisa           :  26
Alexandra      :  27
David          :  27
Johann         :  29
Victor         :  31
Hans           :  31
Ole            :  32
Ulrike         :  34
Erik           :  36
Pia            :  38
Kathrin        :  39
Isolde         :  40
Cäsar          :  43
Daniel         :  47
Tobias         :  47
Florian        :  51
Helene         :  55
Claudia        :  59
Nancy          :  60
Svenja         :  62
Elisabeth      :  65
Zelda          :  66
Benjamin       :  68
Lucy           :  69
Peter          :  70
Isabel         :  71
Felix          :  72
Becky          :  74
Gregor         :  74
Michael        :  82
Julia          :  86
Karl           :  93
Willi          :  94
Susi           :  95
----------Test Controlling-Matching------------
Name           :  Points  ; Name           :  Points
Anton          :  1       ; Susi           :  95
Nina           :  5       ; Willi          :  94
Romeo          :  10      ; Karl           :  93
Markus         :  11      ; Julia          :  86
Gunther        :  11      ; Michael        :  82
```

```
Tanja          :  12      ; Gregor         :  74
Victoria       :  12      ; Becky          :  74
Otto           :  12      ; Felix          :  72
Zoey           :  15      ; Isabel         :  71
Maria          :  15      ; Peter          :  70
Quentin        :  16      ; Lucy           :  69
Moritz         :  19      ; Benjamin       :  68
Xena           :  19      ; Zelda          :  66
Tristan        :  22      ; Elisabeth      :  65
Yvonne         :  23      ; Svenja         :  62
Ronny          :  23      ; Nancy          :  60
Lisa           :  26      ; Claudia        :  59
Alexandra      :  27      ; Helene         :  55
David          :  27      ; Florian        :  51
Johann         :  29      ; Tobias         :  47
Victor         :  31      ; Daniel         :  47
Hans           :  31      ; Cäsar          :  43
Ole            :  32      ; Isolde         :  40
Ulrike         :  34      ; Kathrin        :  39
Erik           :  36      ; Pia            :  38
----------Test Immediate-Matching------------
Name           :  Points  ; Name           :  Points
Anton          :  1       ; Michael        :  82
Nina           :  5       ; Willi          :  94
Romeo          :  10      ; Lucy           :  69
Markus         :  11      ; Felix          :  72
Gunther        :  11      ; Susi           :  95
----------Test Group-Matching------------
true 0
------Group 10------
Anton          :  1
Susi           :  95
Quentin        :  16
Lucy           :  69
Erik           :  36
------Group 9------
Nina           :  5
Willi          :  94
Moritz         :  19
Benjamin       :  68
Pia            :  38
------Group 8------
Romeo          :  10
Karl           :  93
Xena           :  19
Zelda          :  66
Ulrike         :  34
------Group 7------
Markus         :  11
Julia          :  86
Tristan        :  22
Elisabeth      :  65
Kathrin        :  39
------Group 6------
Gunther        :  11
Michael        :  82
Yvonne         :  23
Svenja         :  62
Ole            :  32
```

```
------Group 5------
Tanja          :  12
Gregor         :  74
Ronny          :  23
Nancy          :  60
Isolde         :  40
------Group 4------
Victoria       :  12
Becky          :  74
Lisa           :  26
Claudia        :  59
Hans           :  31
------Group 3------
Otto           :  12
Felix          :  72
Alexandra      :  27
Helene         :  55
Cäsar          :  43
------Group 2------
Zoey           :  15
Isabel         :  71
David          :  27
Florian        :  51
Victor         :  31
------Group 1------
Maria          :  15
Peter          :  70
Johann         :  29
Tobias         :  47
Daniel         :  47
true 2
------Group 12------
Anton          :  1
Susi           :  95
Xena           :  19
Zelda          :  66
Erik           :  36
------Group 11------
Nina           :  5
Willi          :  94
Tristan        :  22
Elisabeth      :  65
Pia            :  38
------Group 10------
Romeo          :  10
Karl           :  93
Yvonne         :  23
Svenja         :  62
------Group 9------
Markus         :  11
Julia          :  86
Ronny          :  23
Nancy          :  60
------Group 8------
Gunther        :  11
Michael        :  82
Lisa           :  26
Claudia        :  59
------Group 7------
```

```
Tanja          :  12
Gregor         :  74
Alexandra      :  27
Helene         :  55
------Group 6------
Victoria       :  12
Becky          :  74
David          :  27
Florian        :  51
------Group 5------
Otto           :  12
Felix          :  72
Johann         :  29
Tobias         :  47
------Group 4------
Zoey           :  15
Isabel         :  71
Victor         :  31
Daniel         :  47
------Group 3------
Maria          :  15
Peter          :  70
Hans           :  31
Cäsar          :  43
------Group 2------
Quentin        :  16
Lucy           :  69
Ole            :  32
Isolde         :  40
------Group 1------
Moritz         :  19
Benjamin       :  68
Ulrike         :  34
Kathrin        :  39

Name              Points
Anton          :  1
Nina           :  5
Romeo          :  10
Markus         :  11
Gunther        :  11
Victoria       :  12
Otto           :  12
Tanja          :  12
Zoey           :  15
Maria          :  15
Quentin        :  16
Moritz         :  19
Xena           :  19
Tristan        :  22
Ronny          :  23
Yvonne         :  23
Lisa           :  26
Alexandra      :  27
David          :  27
Johann         :  29
Hans           :  31
Victor         :  31
Ole            :  32
```

```
Ulrike         :  34
Erik           :  36
Pia            :  38
Kathrin        :  39
Isolde         :  40
Cäsar          :  43
Tobias         :  47
Daniel         :  47
Florian        :  51
Helene         :  55
Claudia        :  59
Nancy          :  60
Svenja         :  62
Elisabeth      :  65
Zelda          :  66
Benjamin       :  68
Lucy           :  69
Peter          :  70
Isabel         :  71
Felix          :  72
Gregor         :  74
Becky          :  74
Michael        :  82
Julia          :  86
Karl           :  93
Willi          :  94
Susi           :  95
----------Test Controlling-Matching------------
Name           :  Points  ; Name           :  Points
Anton          :  1       ; Susi           :  95
Nina           :  5       ; Willi          :  94
Romeo          :  10      ; Karl           :  93
Markus         :  11      ; Julia          :  86
Gunther        :  11      ; Michael        :  82
Victoria       :  12      ; Becky          :  74
Otto           :  12      ; Gregor         :  74
Tanja          :  12      ; Felix          :  72
Zoey           :  15      ; Isabel         :  71
Maria          :  15      ; Peter          :  70
Quentin        :  16      ; Lucy           :  69
Moritz         :  19      ; Benjamin       :  68
Xena           :  19      ; Zelda          :  66
Tristan        :  22      ; Elisabeth      :  65
Ronny          :  23      ; Svenja         :  62
Yvonne         :  23      ; Nancy          :  60
Lisa           :  26      ; Claudia        :  59
Alexandra      :  27      ; Helene         :  55
David          :  27      ; Florian        :  51
Johann         :  29      ; Daniel         :  47
Hans           :  31      ; Tobias         :  47
Victor         :  31      ; Cäsar          :  43
Ole            :  32      ; Isolde         :  40
Ulrike         :  34      ; Kathrin        :  39
Erik           :  36      ; Pia            :  38
----------Test Immediate-Matching------------
Name           :  Points  ; Name           :  Points
Anton          :  1       ; Michael        :  82
Nina           :  5       ; Willi          :  94
Romeo          :  10      ; Lucy           :  69
```

```
Markus         :  11      ; Felix          :  72
Gunther        :  11      ; Susi           :  95
----------Test Group-Matching------------
true 0
------Group 10------
Anton          :  1
Susi           :  95
Quentin        :  16
Lucy           :  69
Erik           :  36
------Group 9------
Nina           :  5
Willi          :  94
Moritz         :  19
Benjamin       :  68
Pia            :  38
------Group 8------
Romeo          :  10
Karl           :  93
Xena           :  19
Zelda          :  66
Ulrike         :  34
------Group 7------
Markus         :  11
Julia          :  86
Tristan        :  22
Elisabeth      :  65
Kathrin        :  39
------Group 6------
Gunther        :  11
Michael        :  82
Ronny          :  23
Svenja         :  62
Ole            :  32
------Group 5------
Victoria       :  12
Becky          :  74
Yvonne         :  23
Nancy          :  60
Isolde         :  40
------Group 4------
Otto           :  12
Gregor         :  74
Lisa           :  26
Claudia        :  59
Victor         :  31
------Group 3------
Tanja          :  12
Felix          :  72
Alexandra      :  27
Helene         :  55
Cäsar          :  43
------Group 2------
Zoey           :  15
Isabel         :  71
David          :  27
Florian        :  51
Hans           :  31
------Group 1------
```

```
Maria          :  15
Peter          :  70
Johann         :  29
Daniel         :  47
Tobias         :  47
```

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe;
die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsinstitution vorgelegt und ist auch noch nicht veröffentlicht worden.

Ort, Datum *Unterschrift*