Fachhochschule Nordwestschweiz

P7: Algoria - Persönliche Lernumgebung

Projektbericht

University of Applied Sciences Northwestern Switzerland - Computer Sciences

Studierender:	Reto Frey
Betreuender Dozent:	Prof. Dr. Christoph Stamm

Revision:	1.0
Datum:	1. März 2012

Zusammenfassung / Abstract

Zusammenfassung

Das Programm *Algoria Worksheet* bietet Studierenden und Dozierenden die Möglichkeit den Computer im Algorithmen und Datenstrukturen Unterricht sinnvoll einzusetzen. Mithilfe einer interaktiven Zeichenfläche, welche Datenstrukturen erkennen kann, können Aufgaben an die Studierenden gestellt werden, die beantwortet und automatisiert überprüft werden können. Die Prüfung erstellt eine Metrik (Anzahl Schritte, welche notwendig sind, um aus einer gegebenen Datenstruktur eine gesuchte zu machen), mit welcher die Ähnlichkeit von Datenstrukturen beschrieben werden kann. Welche Operation dabei wie schwer gewichtet wird, kann nur mit einer fest eingestellten konstanten Zahl konfiguriert werden.

In der vorliegenden Arbeit wird die Erweiterung des genannten Programmes beschrieben. Dies umfasst nebst kleinen Änderungen und Ausmerzen von Fehlern vor allem das Ausführen der automatisierten Lösungsprüfung. Der Prüfungsvorgang wurde generalisiert und grösstmöglich unabhängig von den zu prüfenden Datenstrukturen gemacht. Der nicht generisch zu prüfende Anteil wurde in die jeweilige Datenstruktur verlagert. Die Gewichtung der einzelnen Operationen wurde dynamisiert. Jede Datenstruktur wurde um die Funktion erweitert die verfügbaren Operationen und deren Standardgewicht (konstant oder in Abhängigkeit der Grösse der Datenstruktur) bekannt zu geben. Damit das korrekte Funktionieren der Prüfung innerhalb einer Datenstruktur gewährleistet werden kann und sicher zu stellen, dass jede Datenstruktur definierte Standardoperationen zur Verfügung stellt, wurden Code Contracts verwendet.

Nebst konkreten Implementierungen wurden zwei Konzepte erstellt, welche in einem späteren Projekt Verwendung finden werden. Zum einen wurde ein automatisiertes Vergleichen von mehreren Lösungspaaren mit verschiedenen Einstellungen überprüft. Mithilfe eines solchen automatischen Ablaufs kann der Einfluss der Einstellungen auf den Lösungsprüfer getestet werden. Dies würde einem Dozierenden helfen, die richtigen Gewichtungen für die Operationen, welche beim Erstellen der Metrik verwendet werden, gut zu setzen. Ein zweites Konzept befasst sich mit der Kollaboration unter mehreren Benutzern von *Algoria Worksheet*. Die Kollaboration umfasst sowohl das gleichzeitige Bearbeiten von Fragen, wie auch das Austeilen und Einsammeln von Aufgaben. Eine wichtige Grundvoraussetzung dafür, dass ein gleichzeitiges Bearbeiten möglich ist, ist die Synchronisierung zwischen den verschiedenen Instanzen. Für das Synchronisieren wurden mehrere Vorschläge gemacht.

Abstract

The program *Algoria Worksheet* provides the ability to students and lecturers to use the computer in a reasonable way in the algorithms and datastructures lectures. An interactive draw board, which recognizes datastructures, is used to set a task for a student. The student can now answer the provided question which can be verified in an automatic way. The checking procedure creates a metric value (the number of operations that are used to transform a provided datastructure to a wanted solution), which describes the similarity between datastructures. The impact of an operation to the metric value is a hardcoded constant that can be changed only in the code.

In the present report the extensions to the mentioned program are described. This encloses small changes as well as the fixing of several small bugs and misbehaviors. The most work went into the automatic testing of solutions. The checking procedure has been generalized and made mostly independent from the datastructures to check. Each step that could not be generalized has been moved to the datastructure itself. The weighting of the operations has been made more dynamically. Each

datastructure provides a list of available operations and some default weights for those (constant or even in respect to the length of the datastructure). To secure the correct behavior of the solution checking inside the datastructure as well as the providing of standard operations, code contracts have been used.

Besides the concrete implementations, two concepts have been developed which will be used in a later project. For one thing, an automatic comparing of multiple solution pairs with different settings has been checked. This would provide the possibility for a lecturer to test the right weighting of an operation (that will be used for calculating the metric value) for a certain question. A second concept answers the question, how a collaboration between multiple users of *Algoria Worksheet*. The collaboration includes the simultaneous answering as well as the distributing and the colleting of tasks. One of the basic requirements for a simultaneous working is the synchronization between different instances. For these synchronization tasks multiple suggestions have been made.

Inhaltsverzeichnis

1 Einleitung

Mit dem Programm *Algoria Worksheet* wurde ein Grundstein für das Arbeit mit dem Computer im Algorithmen und Datenstrukturen Unterricht gelegt. Dieser Unterricht ist ein essentieller Bestand der Informatikausbildung an der Fachhochschule Nordwestschweiz. Mithilfe der Administrationsversion des Programmes können Dozierende Aufgabenblätter erstellen und an Studierende verteilen. In einer Studierendenversion können diese Aufgabenblätter bearbeitet werden. Für eine automatisierte Lösungsprüfung wurde ein Konzept erstellt, welches auch im Ansatz implementiert worden ist. Die Erstellung des Programmes sowie das Ausarbeiten des Konzeptes für den Lösungsprüfer erfolgte während der gleichnamigen Bachelor Thesis.

Die Schlüsselfragen, welche in dieser Arbeit bearbeitet wurden, können in zwei Teile aufgespalten werden. Zum einen handelt es sich um einen Praxisteil, welcher sich mit konkreten Implementierungen befasst. Zum anderen wurden zwei konzeptuelle Fragen bearbeitet. Die Praxisaufgaben waren:

- Durchführen und Auswerten eines Usability-Test
- Umsetzen der geforderten Änderungen der Usability-Test-Auswertung
- Wie kann die Korrektheit der Software gewährleistet werden?
- Wie kann der Lösungsprüfer verallgemeinert werden?

Nebst diesen praktischen Arbeiten, wurden zwei Konzeptthemen bearbeitet. Die dazu gehörenden Themen umfassten:

- Wie kann die Kollaboration mit Algoria Worksheet realisiert werden?
- Wie kann das Verhalten des Lösungsprüfers automatisiert getestet werden?

2 Disposition

Die Unterkapitel der Disposition liefern Informationen über die Ausgangslage dieses Projekts und beschreiben die Anforderungen an die Lösung. Im Kapitel 2.2 werden die Fragen aus der Einleitung aufgegriffen und deren Relevanz für die Software aufgezeigt.

2.1 Ausgangslage

Wie bereits im Kapitel 1 angesprochen basiert diese Arbeit auf der Bachelor Thesis "Algoria Worksheet". Wo es sich um Verbesserungen der bereits bestehenden Implementierungen handelt, wird im Kapitel 2.2 beschreiben. Im Folgenden wird die Ausgangslage aus der Aufgabenstellung gezeigt.

> Die Verwendung des Computers im Unterricht ist heute selbstverständlich, doch oft werden Computer häufig nur verwendet, um Power-Point Präsentationen abzuspielen. Dem wollen wir mit einer interaktiven, kollaborativen Lernumgebung für den Algorithmen und Datenstrukturen Unterricht entgegenwirken. Diese Lernumgebung baut auf der Anwendung Algoria auf, welche durch ein natürliches Interface zum Zeichnen von Datenstrukturen hervorsticht. Mittels Skizzenerkennung werden geometrisch beschriebene, aber nicht programmierte Symbole automatisch erkennt und mit vordefinierten Datenstrukturen assoziiert. Die Datenstrukturen und zugehörigen Algorithmen selber werden mit Hilfe einer Plugin-Architektur in Algoria integriert. Gezeichnete Datenstrukturen werden jedoch nicht nur erkannt, sondern im Speicher auch gleich nachgebildet, so dass Algorithmen auf diesen gezeichneten Datenstrukturen ausgeführt und animiert werden können. Algoria ist primär als Einzelplatzanwendung konzipiert worden, mit einem Hauptfokus als Dozenten-SW. Aufbauend auf Algoria ist eine erste Version eines digitalen Arbeitsblattes entstanden, welches Algoria nicht nur in Verbindung mit virtuellen Whiteboards einsetzen lässt, sondern darüber hinaus als persönliche Lernumgebung für den Algorithmen- und Datenstrukturen-Unterricht positioniert[1]

2.2 Anforderungsbeschreibung

Während der Bachelor Thesis wurde bereits ein Usability Test gemacht. Dieser lieferte aber nicht die gewünschten Resultate und wurde nur mit einer der beiden Hauptbenutzergruppen durchgeführt. Aus diesem Grund soll ein weiterer durchgeführt werden. Die Resultate sollen direkt in den Entwicklungsprozess einfliessen.
Auch die Überprüfung der Korrektheit des geschriebenen Codes wurde in der genannten Thesis nur stellenweise berücksichtigt. Während der Weiterentwicklung des Codes soll das Konzept der Code Contracts[2] überprüft und an sinnvollen Stellen eingesetzt werden.
Der während der Bachelor Thesis implementierte Lösungsprüfer wurde aufgrund von Zeitproblemen nicht vollständig fertiggestellt und bedarf einiger Anpassungen. Das Gewichten von Operationen wurde nur im Code ermöglicht und die einzige Datenstruktur, mit welcher der Lösungsprüfer arbeiten konnte, war das Array. Falls in den zu vergleichenden Mengen von Datenstrukturen mehr als jeweils nur eine Datenstruktur vorhanden ist, schlug das Bilden von Paaren (ein Lösungsarray eines Studierenden, mit dem eines Dozierenden), sodass eine möglichst hohe Ähnlichkeit der Mengen entsteht, häufig fehl. Der Lösungsprüfer soll, wo möglich, generisch implementiert. Das Bilden von optimalen Datenstrukturpaaren soll die Ähnlichkeit maximieren.

[1] Aus der Aufgabenstellung[1]
[2] http://research.microsoft.com/en-us/projects/contracts/[2]

Grundsätzlich wird ein Ausmerzen von Fehlern im Programm notwendig. Es wird während des Semesters immer wieder zu kleinen Implementierungen kommen. Diese sollen im Bericht nur kurz aufgelistet werden. Es handelt sich hierbei um die folgende, nicht abschliessende Liste:

- Avalon Dock Bug (das Programm stürzt auf 32-Bit Windows Maschinen beim Herauslösen eines Avalon Dock Fensters ab)
- Visualisieren des Ladens von Aufgaben
- Die Vorschaubilder in der Seitenleiste sollen den Zoomfaktor anhand der enthaltenen Datenstrukturen einstellen
- Drag and Drop Visualisierung soll überarbeitet werden
- ...

Es soll ein Konzept für die Kollaboration mehrere *Algoria Worksheet*-Instanzen erstellt werden. Dieses Konzept soll beschreiben, wie mehrere Studierende gemeinsam eine Aufgabe bearbeiten, Dozierenden den Fortschritt einer Aufgabe im Unterrichtsumfeld überwachen oder Arbeiten verteilt und eingesammelt werden können. Der damit verbundene Synchronisationsaufwand soll konzeptionell beschrieben werden.
Das automatische Testen von Auswirkungen von verschiedenen Einstellungen des Lösungsprüfers auf die erstellten Metriken soll konzeptionell geplant werden. Dafür sollen, so gut möglich, nur bereits erstellte Komponenten verwendet werden.

2.3 Personas

Damit die Änderungen an *Algoria Worksheet* so gut wie möglich auf die Bedürfnisse und Fähigkeiten der Benutzer anzupassen, wurden die Personas aus der Bachelor Thesis übernommen. Es handelt sich hierbei um zwei grundverschiedene Gruppen. Zum einen stellen Dozierende die Gruppe dar, welche die Administrationsversion verwenden. Die Studierendenversion wird primär von Studierenden verwendet. Aufgrund dieser zwei Gruppen wurden die jeweiligen Personas als repräsentative Vertreter erstellt.

2.3.1 Dozierender

Name	Rick Kuhn
Rolle / Gruppe	Dozent für Algorithmen und Datenstrukturen
Einstellung	Nimmt sich Zeit und arbeitet intensiv mit dem Programm. Als Dozierender ist er daran interessiert, dass die Studierenden mit dem Programm arbeiten. Er selbst erstellt Aufgaben und Lösungen und nutzt daher die Software regelmässig im vollen Ausmass. Für ihn sind vor allem die administrativen Funktionen wichtig.
Erwartungen an Algoria	• Die Software soll intuitiv bedienbar sein. • Die Software soll leicht verständlich sein. • Verschiedene Aufgabentypen sollen einfach erfasst werden können (Zeichnen, Multiple Choice, etc.). • Das Programm soll auch die von Studierenden erarbeiteten Lösungen anzeigen können. • Besitzt einen Laptop mit Tabletfunktion, d.h. er kann darauf zeichnen. Daher verwendet er diese Funktion gerne.
Vorkenntnisse	Als Dozent für Algorithmen und Datenstrukturen mit einem Doktor-Titel und einigen Jahren Erfahrung in einer unterrichtenden Form, verfügt er über einige Vorkenntnisse im Bereich Informatik und Didaktik. Er ist es sich gewohnt mit Mircosoft Programmen zu arbeiten und kennt sich daher mit den gängigsten Tastenkürzeln aus. Da er schon einige Aufgaben für Studierende erstellt hat, hat er klare Vorstellungen, wie diese in elektronischer Form aussehen könnten.

Tabelle 2.1: Dozierenden Persona

2.3.2 Studierender

Name	Ulf Poole
Rolle / Gruppe	Student im Bereich Informatik
Einstellung	Hat je nach Belastung wenig Geduld und Interesse an mühsamen Arbeiten. Bevorzugt daher die Arbeit mit einfachen, selbsterklärenden Mitteln. Unterbricht die Arbeit öfters, da er einfach durch andere Dinge abgelenkt wird.
Erwartungen an Algoria	• Die Software soll intuitiv bedienbar sein. • Die Software soll leicht verständlich sein. • Besitzt einen Laptop mit Tabletfunktion, d.h. er kann darauf zeichnen. Daher verwendet er diese Funktion gerne.
Vorkenntnisse	Da Ulf ein Student im Bereich Informatik ist, hat er sehr gute Kenntnisse im Umgang mit Computerprogrammen. Er arbeitet viel mit Microsoft Programmen wie Word oder VisualStudio und kennt daher die Prinzipien von: Toolbars, verschiebbaren Seitenleisten und Menüstrukturen. Er benutzt sehr oft Tastenkürzel wie Ctrl+C/Ctrl+V, da er es sich vom Programmieren gewöhnt ist, selten die Maus in der Hand zu haben.

Tabelle 2.2: Studierenden Persona

2.4 Anwendungsszenarien

Das Programm *Algoria Worksheet* wurde primär für die Verwendung im Algorithmen und Datenstrukturen Unterricht entworfen. Diese Umgebung fordert von der Implementierung in mehreren Anwendungsszenarien zu funktionieren und die benötigten Funktionalitäten zur Verfügung zu stellen. Es handelt sich hierbei um das Vor- und Nachbereiten von Unterrichtslektionen, dem Arbeiten während des Unterrichts und dem Vorbereiten auf Prüfungen. Die Szenarien werden im Folgenden genauer beschrieben.

2.4.1 Vor und Nachbereiten des Unterrichts

In dieser Version von *Algoria Worksheet* ist vorgesehen, dass die Dozierenden Aufgaben in Form von *Algoria Worksheet* Dateien an die Studierenden verteilen. Dies geschieht über eine von der Schule offerierten Methoden, beispielsweise über das Active Directory.

Die Studierenden haben dann die Möglichkeit, diese Aufgabenblätter mit ihrer Kopie von *Algoria Worksheet* zu öffnen und zu lösen. Es ist möglich, die Aufgaben auf jedem beliebigen Computer zu lösen, unter der Voraussetzung, dass *Algoria Worksheet* installiert ist. Die Studierenden sind daher von Zeit und Ort unabhängig.

Um die Aufgaben zu überprüfen, gibt es den Lösungsprüfer in *Algoria Worksheet*. Er zeigt Fehler in der Lösung des Benutzers an und hilft so die korrekte Antwort zu finden. Unklarheiten, welche nicht mit Hilfe des Lösungsprüfers oder *Algoria Worksheet* beseitigt werden können, sollen in den Unterricht verlagert werden. Die Dozierenden können dort konkret auf die Fragen eingehen und zusammen mit den Studierenden die Lösungen in *Algoria Worksheet* besprechen. Die Kontaktstunden sind weiterhin die wichtigsten Teile des Unterrichtes. *Algoria Worksheet* kann aber den Studierenden bei der Aufarbeitung des Stoffes helfen und den Dozierenden Zeit für das Erstellen und Korrigieren von Aufgabenblättern einsparen.

2.4.2 Während des Unterrichts

Algoria Worksheet soll helfen, den Unterricht interaktiver zu gestalten. Die Studierenden erhalten Aufgabenblätter, welche sie mit *Algoria Worksheet* lösen können. Das eingebaute Algoria hilft dabei, die Algorithmen und Datenstrukturen zu verstehen, indem einzelne Schritte aufgezeigt werden können. Da aktuell keine Kommunikation zwischen *Algoria Worksheet* Instanzen möglich ist, wird sich der Einsatz auf einzelne Übungsblöcke beschränken, nach welchen die Aufgaben von den Dozierenden mittels *Algoria Worksheet* besprochen werden können. Es ist vorstellbar, dass sich ein Folgeprojekt mit der Interaktion zwischen mehreren Instanzen von *Algoria Worksheet* beschäftigt. Dadurch könnten die Aufgaben in Unterrichtslektionen in Zusammenarbeit mehrerer Personen gelöst werden.

2.4.3 Prüfungsvorbereitungen

Das wahrscheinlich zutreffendste Szenario findet sich in den Prüfungsvorbereitungen wieder. Viele Studierende werden *Algoria Worksheet* vor den Prüfungen intensiv benutzen und die erhaltenen Aufgabenblätter durcharbeiten. Durch den eingebauten Lösungsprüfer können sie ihre Antworten selbst auf Korrektheit prüfen lassen. Es ist davon auszugehen, dass *Algoria Worksheet* einen grossen Anteil der Prüfungsvorbereitungen ausmacht.

Zur Zeit ist es unwahrscheinlich, dass Prüfungen in *Algoria Worksheet* abgehalten werden können. Diese Möglichkeit soll aber nicht zwangsläufig ausgeschlossen werden. Eventuell wird es in Zukunft möglich sein. Die Voraussetzungen sind von *Algoria Worksheet* gegeben, denn der Lösungsprüfer kann ausgeschaltet werden.

3 Usability Tests

Wie bereits beschrieben, wird die zu erweiternde Software *Algoria Worksheet* von zwei verschiedenen Benutzergruppen verwendet. Diese haben grundverschiedene Ansprüche an das Programm und möchten, dass diesen Ansprüchen Rechnung getragen wird. Da diese Ansprüche nicht nur durch ein Gespräch mit Vertretern aus den Gruppen herausgefunden werden kann, wurde ein Usability Test gemacht. Es wurde jeweils eine Aufgabenstellung erstellt, welche durch die Probanden bearbeiten werden musste. Während des gesamten Ablaufs wurden Notizen gemacht, welche danach ausgewertet und je nach Relevanz im Programm umgesetzt wurden. Im Folgenden werden die zwei Test beschrieben und das daraus gezogene Fazit dargelegt.

3.1 Dozierendenversion

Die Hauptaufgabe der Dozierenden besteht im Erstellen von Aufgaben. Dazu gehört sowohl das Erstellen eines neuen Arbeitsblattes, dem Eintragen von spezifischen Aufgaben, deren Lösungen und dem Abspeichern des Arbeitsblattes. Zusätzlich sollte die Dozierende, welche den Test absolvierte, eine Kopie des Arbeitsblattes wieder öffnen und eine Datenstruktur aus *Algoria Worksheet* kopieren und in Microsoft Word wieder einfügen.[1]
Aufgaben in *Algoria Worksheet* umfassen solche mit konkreten Lösungen als auch derartige, in welchen nur die Ordnung innerhalb einer Datenstruktur relevant ist. Nebst Aufgaben, welche mit einer gezeichneten Datenstruktur beantwortet werden können, ist es auch möglich Multiple Choice Fragen zu stellen. Die Dozierende sollte auch eine solche Aufgabe erstellen.

3.1.1 Fazit

Im Folgenden werden die Notizen, welche während der Durchführung des Usability Tests gemacht wurden, ausgewertet. Allgemein wurden folgende Äusserungen gemacht bzw. wurden die folgenden Punkte bemerkt:

- Die Sprache ist nicht einheitlich. Vor allem die Menu-Kommandos (Öffnen, Speichern, etc...) sind in Deutsch, die restliche Applikation ist in Englisch.
- Die Schriftarten sind nicht einheitlich.
- Nach dem Ende eines Algorithmus wird ein Reset durch den “Stop”-Knopf in den Algo-Controls durchgeführt. Dies ist nicht als intuitiv empfunden worden.
- Nach dem Ende eines Algorithmus bleibt die Datenstruktur zu lange markiert.

Zu den verschiedenen Sprachen ist zu sagen, dass dies durch die verwendeten Application Commands[2] verursacht wird. Diese orientieren sich an der eingestellten Sprache des Betriebssystems.
Die uneinheitlichen Schriftarten wurden so angepasst, dass eine Schriftart den Standard bildet und nur diese verwendet wird.
Die Anmerkungen bezüglich der Selektion der Datenstruktur nach Ablaufen eines Algorithmus und dem Verwenden des “Stop”-Knopf als Reset in den Algo-Controls wurden weitergeleitet, sind aber nicht Teil dieses Projektes.

Im Folgenden die Notizen währen des Eintragens der Aufgaben:

- Die Textboxen für die Aufgabenstellung und den Titel der Aufgabe sind nicht klar als solche zu erkennen.

[1] Dozierenden Aufgaben Set[3]
[2] http://msdn.microsoft.com/en-us/library/system.windows.input.applicationcommands.aspx[4]

- In der Rich-Text-Editor-Toolbar sind die Knöpfe für Fett / Kursiv und Unterstreichen abgeschnitten.
- Die Rich-Text-Editor-Toolbar ist zu weit von der eigentlichen Textbox entfernt.
- Die Kopieren / Ausschneiden und Einfügen-Knöpfe sollten als inaktiv angezeigt werden, wenn der Fokus nicht auf der Algoria Zeichenfläche oder der Rich-Text-Editor-Box ruht.
- Der Punkt, welcher den Stift im Scribble-Modus darstellt, ist zu klein.
- Die Toggle-Knöpfe für das Wechseln zwischen Ausgangslage und Musterlösung sind zu wenig intuitiv.
- Der Text zum Hinzufügen einer neuen Multiple Choice Antwort ist zu weit vom Knopf weg.
- Die Buchstaben 'C' und 'F' für das Anzeigen einer richtigen oder falschen Multiple Choice Antwort sind nicht intuitiv.

Damit die Textboxen für die Aufgabenstellung und den Titel einfach zu erkennen sind, wurden diese mit Hintergrundtexten versehen, welche den Dozierenden darauf aufmerksam machen, dass noch ein Titel oder eine Aufgabenstellung eingetragen werden sollte. Dieser Text verschwindet solange die Box aktiv ist und wenn sich ein Text darin befindet.
Die abgeschnittenen Bilder der Knöpfe für Fett / Kursiv und Unterstreichen wurden so gesetzt, dass diese ganz zu sehen sind.
Die Rich-Text-Editor-Toolbar konnte aus Platzgründen nicht weiter zu der betreffenden Textbox verschoben werden. Dass die Kopieren / Ausschneiden und Einfügen-Knöpfe in der ganzen Applikation ihre Verwendung finden, ist ein zusätzliches Gegenargument gegen das Verschieben der Toolbar.
Die Knöpfe für das Kopieren, Einfügen und Ausschneiden so anzupassen, dass diese inaktiv erscheinen, wenn der Fokus nicht auf der Algoria Zeichenfläche oder der Rich-Text-Editor-Box ruht, gestaltete sich als schwierig. Aufgrund des schwierigen Managements des Fokus im Windows Presentation Framework[3] wurde dieser Änderungsvorschlag nicht umgesetzt. Dies könnte in einem späteren Projekt angegangen werden.
Der Punkt im Scribble-Modus betrifft die Algoria Zeichenfläche und wurde an die dafür zuständigen Personen weitergeleitet.
Um den Wechsel zwischen Ausgangslage und Musterlösung zu erleichtern, wurde eine zusätzliche Visualisierung eingebaut, welche dem Benutzer anzeigt, in welcher der beiden Zeichenflächen gearbeitet wird. Dazu wurden den beiden Zeichenflächen individuelle Farben zugeordnet. Die Toggle-Buttons wurden analog zu den Zeichenflächen eingefärbt.
Der Text neben dem Knopf für das Hinzufügen einer Multiple Choice Antwort wurde direkt neben den Knopf verschoben.
Für eine bessere Visualisierung, welche Multiple Choice Antworten richtig und welche falsch sind, wurde der Buchstaben 'C' durch ein grünes Häkchen und die 'F' durch ein rotes Kreuz ersetzt.

Im Allgemeinen wurde festgestellt, dass das Erstellen von Aufgaben und das Eintragen von Lösungen und Ausgangslagen intuitiv und ohne Hilfestellung vollzogen wurde. Auch das Erstellen einer Multiple Choice Aufgabe mit mehreren Antwortmöglichkeiten wurde intuitiv erledigt.

3.2 Studierendenversion

Studierende haben nicht die Möglichkeit Aufgaben zu erstellen oder die Aufgabentexte zu verändern. Sie beantworten die Fragen in den Aufgaben auf dem dafür vorgesehenen Weg. Dies kann das Zeichnen einer Datenstruktur sein oder das Auswählen der korrekten Multiple Choice Antwort. Im durchgeführten Test musste der Proband das durch die Dozierende erstellte Arbeitsblatt öffnen und beantworten.

[3] http://windowsclient.net/wpf/[5]

Alle drei Fragen waren zu beantworten und das beantwortete Arbeitsblatt danach wieder zu speichern.[4] Falls bei einer Aufgabe nicht auf Anhieb die richtige Lösung gefunden werden konnte, stand es dem Studierenden frei, den Lösungsprüfer zu benutzen und seine bereits erarbeitete Lösung testen zu lassen.

3.2.1 Fazit

Die Notizen, welche während des Usability Tests mit dem Studierenden gemacht wurde, sind im Folgenden aufgelistet:

- Das Sidepanel wurde intuitiv für das Wechseln zwischen Aufgaben verwendet.
- Der Knopf um den Lösungsprüfer zu starten, wurde in der Toolbar der Algoria Zeichenfläche gesucht.
- Sowohl Öffnen als auch Speichern des Arbeitsblattes funktionierte gut.
- Die Vorschläge des Lösungsprüfers konnten als Hilfestellung verwendet werden.

Der einzige Punkt, welcher als nicht positiv empfunden wurde, ist die Position des Startknopfs für den Lösungsprüfer. Die Verschiebung des Knopfes wurde besprochen und abgelehnt. Positiv zu erwähnen ist, dass der Studierende mit dem Lösungsprüfer sehr gut arbeiten konnte. Die angezeigten Meldungen wurden verstanden und konnten so umgesetzt werden, dass die Lösung immer näher an die Musterlösung heran geführt werden konnte.

[4] Studierenden Aufgaben Set[6]

4 Algoria Worksheet Testset

Damit die Lösungsprüfungsfunktionen von *Algoria Worksheet* stetig überprüft werden können, bedarf es einem Testset. Dieses Testset beschreibt eine Auswahl von Aufgaben, welche alle Möglichkeiten des Lösungsprüfers ausnutzen. Daraus folgt, dass ein derartiges Testset die Bereiche *Konkrete Lösung*, *Ordnungs-Lösung* und *Multiple Choice* umfassen muss. Diese Teilbereiche werden im Folgenden beschrieben, sowie das geforderte Verhalten aufgezeigt.

4.1 Aufgaben mit konkreter Lösung

Konkrete Lösungen sind ein erster Teil der Funktionalität des Lösungsprüfers. In dieser Konfiguration müssen zwei Lösungen nicht nur in der Länge der Datenstrukturen übereinstimmen, sondern müssen auch genau die gleichen Elemente an der gleichen Stelle stehen. Die Tabelle 4.1 listet die erstellten Testaufgaben für konkrete Lösungen auf.

Titel	Inhalt	Kostentypen
Ein Lösungsarray 1	[3, 5, 7, 1, 1, 4]	Add element: Linear 1 Remove element: Linear 1 Set value: Constant 2 Swap: Constant 1
Ein Lösungsarray 2	[3, 7, 1, 3]	Add element: Constant 1 Remove element: Constant 1 Set value: Quadratical 2 Swap: Constant 1
Zwei Lösungsarrays	[3, 5, 7, 1, 1, 4] [1, 7, 3, 3]	Add element: Linear 1 Remove element: Linear 1 Set value: Logarithmic 1 Swap: Constant 1
Vier Lösungsarrays	[2] [6, 2] [9, 11, 3, 2] [5, 2, 4, 1, 1]	Add element: Linear 1 Remove element: Linear 1 Set value: Constant 2 Swap: Constant 1
Kein Lösungsarray	-	-

Tabelle 4.1: Aufgaben für konkrete Lösungen

4.2 Aufgaben mit Ordnung

Nebst dem konkreten Vergleichen von zwei Lösungen ist auch das Auffinden und Abgleichen von Ordnungen innerhalb einer Datenstruktur eine Möglichkeit, welche der Lösungsprüfer bietet. Die Tabelle 4.2 listet die erstellten Testaufgaben für Lösungen basierend auf Ordnung auf.

Titel	Inhalt	Kostentypen
Ein Array aufsteigend	[1, 2, 3, 4, 5, 6]	Add element: Linear 1 Remove element: Linear 1 Set value: Constant 2 Swap: Constant 1
Ein Array absteigend	[6, 5, 4, 3, 2, 1]	Add element: Constant 1 Remove element: Constant 1 Set value: Linear 2 Swap: Constant 1
Ein Array ohne Ordnung	[1, 2, 3, 2, 1, 0]	Add element: Linear 1 Remove element: Linear 1 Set value: Constant 2 Swap: Constant 1
Kein Lösungsarray	-	-
Zwei Lösungsarray	[1, 2, 3, 4, 5, 6] [4, 3, 2, 1]	Add element: Linear 1 Remove element: Linear 1 Set value: Constant Swap: Constant 1

Tabelle 4.2: Aufgaben für geordnete Lösungen

4.3 Multiple Choice Aufgaben

Der Lösungsprüfer kann nebst dem Abgleichen von Datenstrukturen mit der geforderten Lösung auch Multiple Choice Aufgaben auf Korrektheit überprüfen. Die nachfolgende Tabelle 4.3 gibt einen Überblick über die Multiple Choice Aufgaben im Testset.

Titel	Antworten	Antwortmodus
Eine Lösung 1	Falsch Richtig Falsch Falsch	Eine korrekte Antwort
Eine Lösung 2	Falsch Falsch Falsch Falsch	Eine korrekte Antwort
Mehrere Lösungen 1	Falsch Richtig Falsch Falsch	Mehrere korrekte Antworten
Mehrere Lösungen 2	Richtig Richtig Richtig Richtig	Mehrere korrekte Antworten
Mehrere Lösungen 3	Falsch Falsch Falsch Falsch	Mehrere korrekte Antworten

Tabelle 4.3: Aufgaben für Multiple Choice

5 Programm

Die folgenden Unterkapitel beschäftigen sich mit den Änderungen, welche am Programm *Algoria Worksheet* gemacht wurden. Teilweise basierten die Änderungen auf bereits bestehenden Funktionalitäten (vornehmlich das Ausmerzen von Fehlern), andere wurden von Grund auf neu implementiert. Die weitgehendste Änderung, das Anpassen des Lösungsprüfers, wird im Kapitel 6 behandelt.

5.1 Visualisieren des Ladevorgangs

Beim Laden eines Arbeitsblattes können, je nach Leistungsfähigkeit des verwendeten Computers, mehrere Sekunden verstreichen. Der Benutzer verfügt über keine Informationen zum Stand des Ladevorgangs, so entsteht der Eindruck, das Programm sei abgestürzt. Damit dieser Eindruck nicht entsteht, müssen konstant Aktualisierungen ersichtlich sein. Dies geschieht in vielen Programm mittels einem Fortschrittsbalken. Dieser kann sich konstant bewegen, um den Arbeitsprozess anzuzeigen, oder sich stetig füllen, um auch gleich noch die Information der verbleibenden Zeit bis zum Fertigstellen der Aktion zu visualisieren.
Der Einsatz eines Fortschrittsbalken wäre auch in *Algoria Worksheet* eine Möglichkeit gewesen. Durch die Seitenleiste, welche alle enthaltenen Aufgaben anzeigt, bestand jedoch die Möglichkeit einer "natürlichen" Fortschrittsanzeige. Dieser Ansatz wurde auch gewählt und implementiert. Die Aufgaben werden nacheinander geladen und jeweils in der Seitenleiste angezeigt. Durch das stetige Anzeigen und parallele Laden von Aufgaben entsteht die Gefahr, dass eine Änderung an einer noch nicht vollständig geladenen Aufgabe gemacht wird, was zu einem Fehler führen könnte. Damit dieser Problematik entgegen gewirkt werden kann, wird die gesamte Benutzeroberfläche während dem Ladevorgang inaktiv gesetzt und reagiert somit nicht auf Benutzerinteraktionen.
Das stetige Hinzufügen von Aufgaben in Datenablage der Seitenleiste führte jedoch nicht zu einem Aktualisieren des Standes auf der Benutzeroberfläche. Sobald jedoch der Ladevorgang fertiggestellt wurde, waren alle Aufgaben ersichtlich. Das Problem wurde im Aktualisierungsvorgang der Benutzeroberfläche lokalisiert und konnte mithilfe eines *DispatcherFrame*-Objektes gelöst werden. Die Klassen *SidepanelView* und *SidepanelAdminView* verfügen jeweils über einen Dispacher, welche das Aktualisieren der Benutzeroberfläche handhabt. In den View-Klassen wurde der *Update*-Methode jeweils der folgende Codeblock eingefügt.

```
1  var frame = new DispatcherFrame();
2  this.Dispatcher.BeginInvoke(
3      DispatcherPriority.Render,
4      new DispatcherOperationCallback(
5          delegate
6              {
7                  frame.Continue = false;
8                  return null;
9              }),
10     null);
11 Dispatcher.PushFrame(frame);
```

Codeausschnitt 5.1: Manuelles Aktualisieren der Benutzeroberfläche

Der Codeausschnitt 5.1 bewirkt, dass die *Update*-Methode ein unmittelbares Aktualisieren der Benutzeroberfläche zur Folge hat. Ansonsten wird diese Aktualisierung erst zu einem späteren Zeitpunkt durchgeführt, welcher sich nach dem Laden des gesamten Arbeitsblattes befindet.

5.2 Vorschaubilder in der Seitenleiste

Durch das Hinzufügen von Scrollbars für die Algoria Engine, wurde es möglich, eine grössere Zeichenfläche zur Verfügung zu stellen. Vor dieser Anpassung hatte die Zeichenfläche eine fixe Grösse, welche für die Vorschaubilder in der Seitenleiste einfach herunter skaliert wurde. Ohne die Grössenbeschränkung hat die Zeichenfläche ein Mass von 8000 x 8000 Pixel. Dies wiederum auf die kleinen Vorschaubilder zu skalieren, würde keine Übersicht mehr zulassen. Auch das Kopieren einer Datenstruktur für ein Einfügen in Word, was dem Konvertieren der gezeigten Datenstruktur in ein Bildformat entspricht, orientierte sich an der alten, eingeschränkten Grösse der Algoria Engine.
Ein Erstellen eines vernünftigen Vorschaubildes bedarf dem Wissen der umschliessenden Box aller Datenstrukturen, der sogenannten Bounding-Box. Dasselbe Konzept kann auch für das Kopieren mehrerer Datenstrukturen für Word verwendet werden. In diesem Falle wird nur die Bounding-Box der selektierten Datenstrukturen berechnet und der Inhalt dieser in einem Bildformat abgespeichert. Im Folgenden wird aufgezeigt, wie diese Box berechnet wird.

```
1 var bounds = AlgoEngine.DatastructureHandles.Select(
2     h => LayoutHelper.GetBounds(h.Datastructure.Model));
3
4 var view = bounds.FirstOrDefault();
5 Rect boundingBox = bounds.Aggregate(view, (v, c) =>
6                                                 {
7                                                     v.Union(c);
8                                                     return v;
9                                                 });
```

Codeausschnitt 5.2: Berechnung der Bounding-Box

Der Codeausschnitt 5.2 benutzt den *LayoutHelper* von Algoria. Dieser liefert den Umriss eines Objektes auf der Zeichenfläche, was für jede Datenstruktur auf der Zeichenfläche gemacht wird. Wenn nun nur selektierte Datenstrukturen für die Berechnung der Bounding-Box berücksichtigt werden sollen, kann dies mit folgender Codezeile erreicht werden:

```
1 var bounds = (from dsHandle in AlgoEngine.DatastructureHandles
2               where dsHandle.IsDatastructureSelected
3               select LayoutHelper.GetBounds(dsHandle.Datastructure.Model));
```

Codeausschnitt 5.3: Bounding-Box für selektierte Datenstrukturen

Das Ersetzen der ersten Codezeile in 5.2 durch diejenige im Codeausschnitt 5.3 führt dazu, dass nur selektierte Datenstrukturen in der Variablen *bounds* abgespeichert werden. Anhand der vorhandenen Umrisse in *bounds* wird die Bounding-Box berechnet.

Wie ein Vorschaubild mithilfe eines *ImagesBrush* gemacht und das Bitmap-Bild für die Zwischenablage erstellt wird, wurde bereits in der Bachelor Thesis zu *Algoria Worksheet* gezeigt.

5.3 Allgemeine Änderungen

Die nachfolgenden Unterkapitel enthalten die zahlreichen kleinen Änderungen, welche gemacht wurden. Für jeden der Bereiche wird kurz die Lösung beschrieben, auf eine weiterführende Beschreibung wird hier verzichtet, damit die geänderten Codestellen gefunden werden können, werden die betroffenen Klassen angegeben. Die Änderungen, welche bereits im Kapitel 3 beschrieben wurden, werden hier nicht noch einmal aufgeführt.

Avalon Dock

Avalon Dock[1] wurde für das freie Herausziehen und Verschieben von Sektionen auf der Benutzeroberfläche (ähnlich Visual Studio 2010) verwendet. Das Herauslösen der Algoria Engine führte jedoch unter 32-Bit zu einem Programmabsturz. Das Problem konnte mit einem *ResizingPanel*, welches die Algoria Engine (beinhaltet in einer *DocumentPane*) umschliesst, gelöst werden.

Betroffene Klassen : *AdminWorksheetShell*, *WorksheetShell*

Algoria Engine Scrollbar

Um die beschränkte Zeichenfläche der Algoria Engines wieder zu erweitern, wurden die Restriktionen in der Grösse wieder aufgehoben und der Zeichenfläche Scrollbars in horizontale und vertikale Richtung gegeben.

Betroffene Klassen : *AEngine*

Drag und Drop

Der Drag und Drop-Vorgang in der Seitenleiste löste die Animation von sehr grossen Boxen aus, welche die möglichen Einfügepunkte einer zu verschiebenden Aufgabe anzeigten. Diese Boxen wurden stark verkleinert und in einer Powerpoint ähnlichen Art umgesetzt. Damit das selektierte Element der Seitenleiste besser zu erkennen ist, wurde ein dickerer und andersfarbiger Rahmen verwendet.

Betroffene Klassen : *SidepanelAdminView*

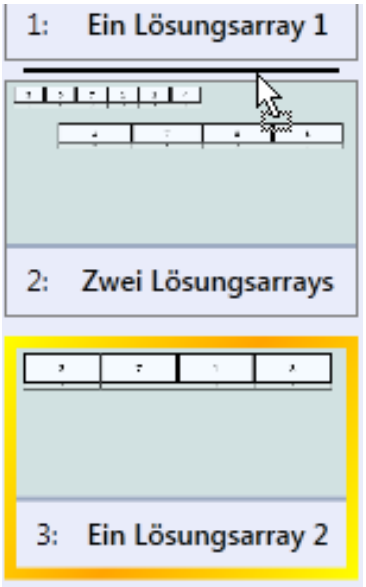

Abbildung 5.1: Selektion und Verschieben einer Aufgabe

[1] http://avalondock.codeplex.com/[7]

Seitenleiste Kopieren und Einfügen

Das Kopieren und Verschieben von Aufgaben innerhalb eines Arbeitsblattes mittels Drag und Drop wurde von *Algoria Worksheet* bereits gegeben. Diese Aktionen wurden so erweitert, dass die Zwischenablage genutzt wird und die Standrad-Tastenkürzel für Kopieren, Einfügen und Ausschneiden verwendet werden können. Das Einfügen einer kopierten Aufgabe für eine andere Applikation als *Algoria Worksheet*, wie es im Falle der Datenstrukturen gemacht wurde, wurde nicht implementiert.

Betroffene Klassen : *AdminWorksheetShell*

Warndialog vor Schliessen

Schliesst ein Benutzer ein Programm ohne die zuvor gemachten Änderungen an einer Datei zu speichern, kann dies zum Verlust von einer grosser Menge Arbeit führen. Um diesen Fall zu verhindern, setzen viele Programme auf eine Meldung, welche den Benutzer daran erinnert, dass gemachte Änderungen noch nicht gespeichert wurden. Dieser Ansatz wurde auch in *Algoria Worksheet* implementiert. Der erscheinende Dialog ist auf der Abbildung 5.2 zu sehen.

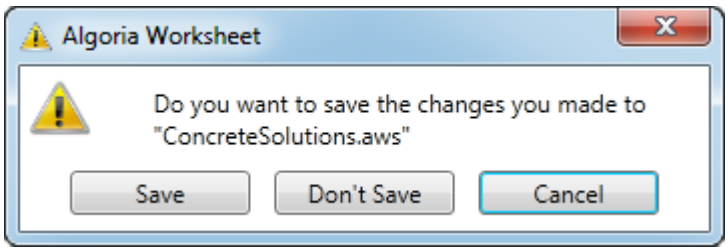

Abbildung 5.2: Dialog für nicht gespeicherte Änderungen

Betroffene Klassen : *SaveChangesDialog*, *AdminWorksheetShell*, *WorksheetShell*, *Controller*

Validierungsdialog

Wenn ein Arbeitsblatt mehr als zwei oder drei Aufgaben enthält, können Dozierende schnell den Überblick verlieren, in welchen Aufgaben sie bereits Lösung, Titel oder Aufgabentext eingetragen haben. Damit eine schnelle Überprüfung möglich wird, wurde eine Validierungsmethode implementiert, welche die fehlenden Elemente zurückliefert. Für jedes fehlende Element wird ein Eintrag in einem Validierungsdialog hinzugefügt. Die Abbildung 5.3 zeigt das Resultat der Validierung für ein Arbeitsblatt, welches Fehler aufweist. Zum Beispiel beinhaltet es eine Aufgabe ohne Titel, eine ohne eingetragene Lösung und eine mit Multiple Choice Antworten, welche keinen Text beinhalten. Im Gegensatz dazu zeigt Abbildung 5.4 die Validierung eines Arbeitsblattes ohne Fehler.

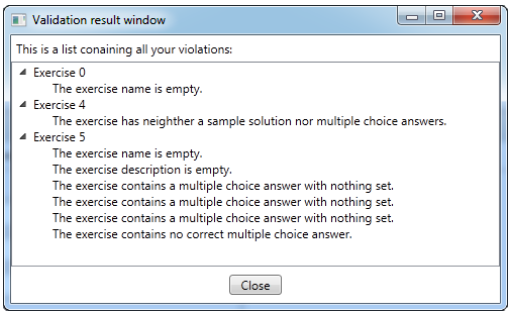

Abbildung 5.3: Validierungsdialog mit Fehlern

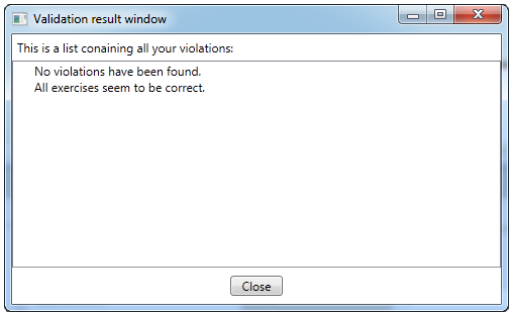

Abbildung 5.4: Validierungsdialog ohne Fehler

Betroffene Klassen : *ValidationErrorWindow*, *AdminWorksheetShell*

6 Lösungsprüfer

Ein wichtiger Teil des Programmes *Algoria Worksheet* macht der Lösungsprüfer aus, welcher den Studierenden die Möglichkeit gibt, ihre Lösung gegen die Musterlösung des Dozierenden zu vergleichen. Dies führt zu einer unmittelbaren Rückmeldung. Diese gibt Aufschluss darüber, wie ähnlich sich die Studierendenlösung und die Musterlösung sind. Während der Bachelor Thesis, in der das Programm erstellt wurde, war der Lösungsprüfer eine konzeptionelle Arbeit. Dieses Konzept wurde nach Abschluss des Projektes noch implementiert, um das Funktionieren des Konzeptes in einer Live-Demonstration aufzuzeigen. Durch die knappe Zeit zwischen dem Abschluss des Projektes und der Präsentation kam es zu einer Implementation, welche noch nicht völlig ausgereift war. In den folgenden Unterkapiteln werden die Verbesserungen am Lösungsprüfer beschrieben. Diese umfassen sowohl die Überarbeitung der Implementierung, dass diese in die Model-View-ViewModel (MVVM)[1] Umgebung passt, die Möglichkeit auch Nicht-Array-Datenstrukturen zu überprüfen, als auch die Implementation von Lösungsprüfungsoptionen.

6.1 Überarbeitung MVVM

Wie bereits angemerkt, entsprach die bisherige Implementierung nicht dem MVVM-Standard. So enthielt die Administrationsversion keine View für den Lösungsprüfer und die Views wurden nicht über die *VersionController*-Klasse abgeholt. Für die einfachere Darstellung des Klassendiagramms wurden in der Abbildung 6.1 nur diejenigen Klassen aufgelistet, welche nach dem Überarbeitung für MVVM für den Lösungsprüfer relevant sind. Ein Klassendiagramm mit den restlichen Klassen ist in der Bachelor Thesis zu finden.

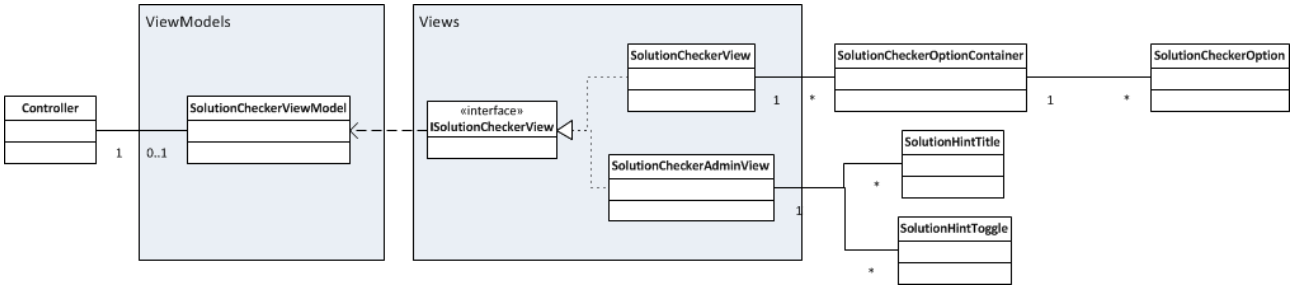

Abbildung 6.1: Klassendiagramm für den Lösungsprüfer

Zusätzlich wurde die *ExerciseEngineContainer*-Klasse mit Feldern für die Daten der Lösungsprüfer-Nachrichten erweitert. Mithilfe dieser Felder können Nachrichten, welche vom Lösungsprüfer für eine Aufgabe erstellt wurden, zwischengespeichert und beim erneuten Laden der Aufgabe angezeigt werden. Die Lösungsprüfungsoptionen werden direkt in der *ExerciseDs*-Klasse gespeichert.

6.2 Verallgemeinerung des Lösungsprüfers

Die bisherige Implementation des Lösungsprüfers beschäftigte sich nur mit Arrays. Andere Datenstrukturen liessen sich nicht überprüfen, sondern führten zum Abbruch. Jede Datenstruktur hat spezifische Aspekte, welche vom Lösungsprüfer jeweils einzeln geprüft werden müssen. Beispielsweise unterscheidet sich das Abgleichen der Reihenfolge innerhalb eines Baums grundlegend von derselben Operation in einem Array. Dies ist ein Teil, welcher für jede Datenstruktur gesondert implementiert werden

[1] http://msdn.microsoft.com/en-us/magazine/dd419663.aspx[8]

muss. Aspekte, welche für alle Datenstrukturen gleich sind, können aber abstrahiert und allgemein implementiert werden. Ein Längenvergleich, oder das Überprüfen, welche gemeinsamen Elemente in zwei Datenstrukturen vorhanden sind, kann generisch gelöst werden. Grundvoraussetzung dafür ist, dass Datenstrukturen in Algoria über eine gemeinsame Schnittstelle angesprochen werden können. Mit der Einführung der abstrakten Klasse *DatastructureElement* im Interface *IDatastructure* wurde diese Möglichkeit gegeben. Abbildung 6.2 zeigt das Ablaufdiagramm des ersten Schrittes im Lösungsprüfer.

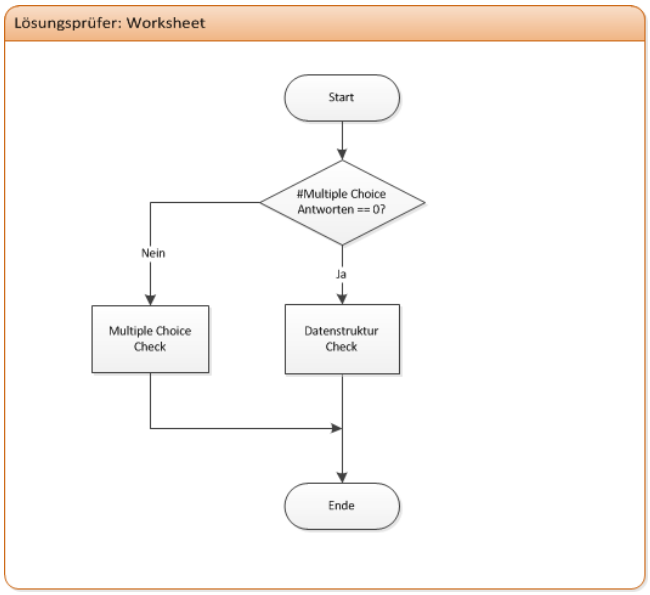

Abbildung 6.2: Initialisierung des Lösungsprüfers

Zu Beginn des Lösungsprüfers überprüft der Controller, wie viele Multiple Choice Antworten vorhanden sind. Sollte es keine Multiple Choice Antwort in der Aufgabe haben, wird ein Datenstruktur Check durchgeführt. Dieser wird in den folgenden Abschnitten behandelt. Sollten Multiple Choice Antworten vorhanden sein, wird ein Multiple Choice Check ausgeführt. Dieser wird im Kapitel 6.4 behandelt.

Abbildung 6.3 auf Seite 24 zeigt den weiteren Ablauf Datenstruktur Checks. In einem ersten Schritt wird die Anzahl der jeweils vorkommenden Datenstrukturen innerhalb der Musterlösung und der Studierendenlösung gezählt. Meldungen für fehlende und überzählige Datenstrukturen in der Studierendenlösung werden in der Solution-Hint-Liste hinzugefügt. Die Anzahl der überzähligen Datenstrukturen wird zwischengespeichert, da für die Datenstrukturen mit den schlechtesten Ähnlichkeitsmassen zum Schluss eine Meldung generiert wird, welche dem Studierenden den Tipp gibt, diese zu löschen.

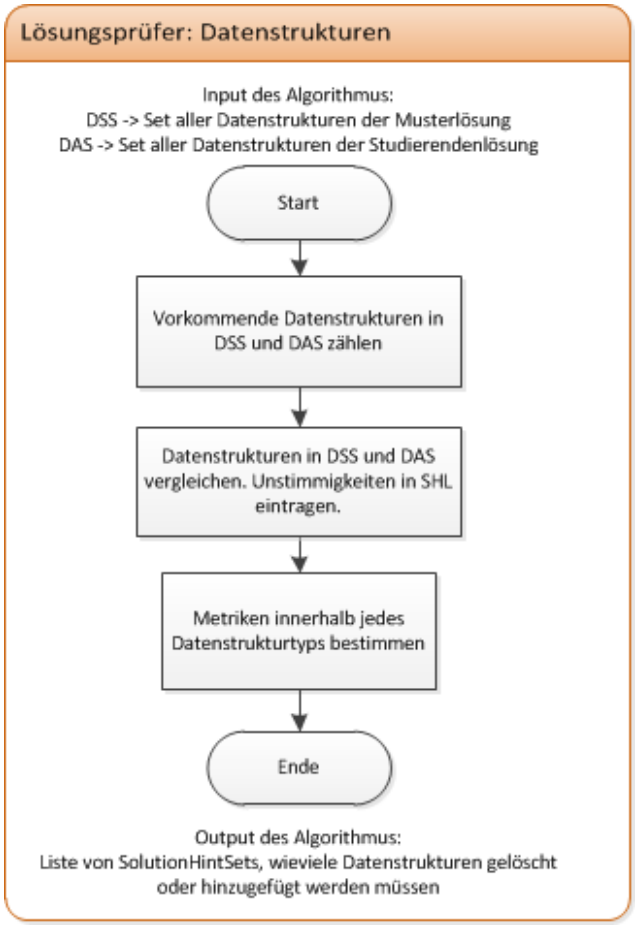

Abbildung 6.3: Datenstrukturen Check Schritt 1

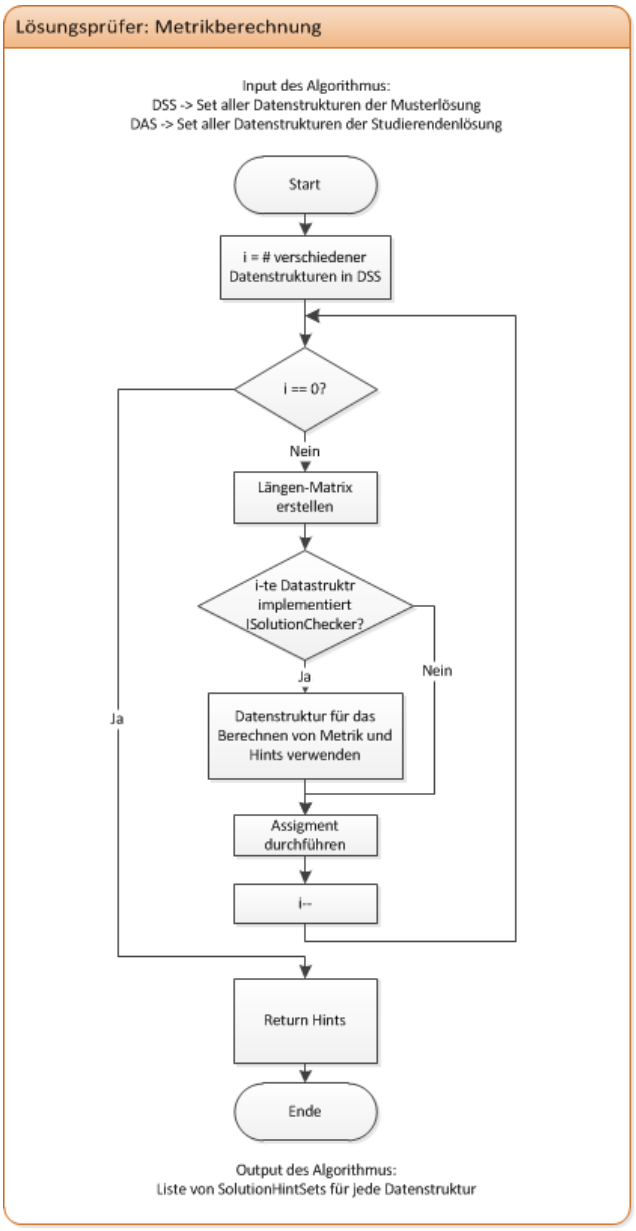

Abbildung 6.4: Datenstrukturen Check Schritt 2

Der Ablauf der Prüfung pro Datenstruktur ist in Abbildung 6.4 zu sehen. Für jeden Datenstrukturtyp wird zu Beginn eine Längen-Matrix erstellt. Die Längen-Matrix wird im Kapitel 6.2.1 behandelt. Danach wird überprüft, ob der jeweilige Datenstrukturtyp das Interface *ISolutionCheckable* implementiert. Das Interface ist im Codeausschnitt 6.1 aufgelistet.

```
1 public interface ISolutionCheckable
2 {
3     ...
4     IEnumerable<Action> ActionsAvailableList { get; }
5     ...
6     Tuple<double, double, IEnumerable<Hint>> CheckForConcreteNumbers(
7         IDatastructure<DatastructureElement> sampleDs, DatastructureHandle
              studentDs, Metric metric);
8     ...
9     Tuple<double, double, IEnumerable<Hint>> CheckForOrder(
```

```
10          IDatastructure<DatastructureElement> sampleDs, DatastructureHandle
                studentDs, Metric metric);
11  }
```

Codeausschnitt 6.1: ISolutionCheckable - Interface

Das *ActionsAvailableList* Feld wird im Kapitel 6.3.1 behandelt. Die beiden Methoden *CheckForConcreteNumbers* und *CheckForOrder* vergleichen zwei gegebene Datenstrukturen vom gleichen Typ auf konkreten Inhalt bzw. gleiche Ordnung. Zurückgeliefert wird ein Tupel, welches den erreichten Metrik-Wert, den maximalen Metrik-Wert für die Musterlösung und eine Liste von Hinweisen enthält. Wenn eine Musterlösung n Elemente enthält, besteht der maximale Metrik-Wert aus n-Additions- und n-SetValue-Operationen zusammen.

Für alle Datenstrukturen, welche das *ISolutionCheckable* implementieren, wird die Längen Matrix mit den zusätzlichen Metrik-Werten aus den datenstrukturtypischen Vergleichmethoden erweitert. Damit jetzt für jede Datenstruktur aus der Studierendenlösung diejenige aus der Dozierendenlösung gefunden wird, wurde das Linear Assigment Problem für den Lösungsprüfer angepasst. In Kapitel 6.2.2 wird dies behandelt.

Basierend auf dem Resultat des Linear Assigment werden die Meldungen generiert, welche Datenstruktur wie angepasst werden muss, um möglichst nahe an die Musterlösung zu kommen. Dies beinhaltet nicht nur die Tipps, die Datenstrukturen um Elemente zu erweitern, zu kürzen oder deren Position zu verschieben, sondern auch welche Datenstrukturen gelöscht werden sollten, oder von welchem Typ noch Datenstrukturen fehlen.

6.2.1 Längen-Matrix

Durch die Möglichkeit, Datenstrukturelemente durch das *IDatastructure*-Interface unabhängig vom Typ der Datenstruktur anzusprechen, können die Längen der Datenstrukturen allgemein verglichen werden. Damit zum Abschluss des Lösungsprüfers ein Linear Assingment Problem für eine optimale Zuweisung von Datenstrukturen in der Studierenden- sowohl auch Musterlösung angewandt werden kann, wird eine Matrix erstellt, welche die jeweiligen Längenunterschiede paarweise auflistet. Zusätzlich wird, sollte ein Datenstruktur-Typ das *ISolutionCheckable*-Interface implementieren und eine Aufgabe auf eine konkrete Lösung überprüft werden, das Vorkommen der Inhalte überprüft. Eine Längen-Matrix, in welcher die gleiche Anzahl Datenstrukturen eines Typs in Studierenden- und Musterlösung enthalten ist, ist in Tabelle 6.1 gezeigt.

	S1	S2	S3	S4
M1	1	4	2	6
M2	2	1	3	4
M3	6	7	3	1
M4	4	2	8	5

Tabelle 6.1: Längen-Matrix bei gleicher Anzahl Datenstrukturen

Die Bezeichnung M1 bezieht sich auf die erste Datenstruktur der Musterlösung. Analog ist S1 die erste Datenstruktur der Studierendenlösung. Die Zelle S1, M1 enthält den Metrik-Wert, wenn beide Datenstrukturen verglichen werden. Für das Anwenden des Linear Assignment Problems ist es notwendig, dass die Längen-Matrix quadratisch ist. Die Tabellen 6.2 und 6.3 zeigen, wie eine Matrix aufgebaut wird, wenn eine Musterlösung mehrere Datenstrukturen beinhaltet als die Studierendenlösung, bzw. die umgekehrte Situation angetroffen wird.

	S1	S2	NV	NV
M1	1	4	-1	-1
M2	2	1	-1	-1
M3	6	7	-1	-1
M4	4	2	-1	-1

Tabelle 6.2: Längen-Matrix bei mehr Musterlösungsdatenstrukturen

Damit auch eine quadratische Matrix erstellt werden kann, wenn zu wenige Studierendendatenstrukturen eingetragen wurden, wird die Matrix mit NV-Spalten gefüllt. NV steht für "Nicht-Vorhanden". Der Vergleich zwischen einer Musterlösungsdatenstruktur und einer Nicht-Vorhandenen liefert immer das Resultat -1. Wird eine Musterlösung beim Linear Assigment einer Nicht-Vorhandenen Datenstruktur zugewiesen, bedeutet dies, dass der Studierende die Datenstruktur in der Musterlösung noch nicht gezeichnet hat, bzw. die zugewiesene Musterlösungsdatenstruktur keiner der Studierendenlösungsdatenstrukturen genügend ähnlich ist.

	S1	S2	S3	S4
M1	1	4	2	6
M2	2	1	3	4
EN	-1	-1	-1	-1
EN	-1	-1	-1	-1

Tabelle 6.3: Längen-Matrix bei mehr Studierendenlösungsdatenstrukturen

Sollte die Studierendenlösung über eine grössere Anzahl Datenstrukturen verfügen als die Musterlösung, wird dasselbe Verfahren angewendet wie zuvor. In der Tabelle 6.3 werden zusätzliche Zeilen mit der Bezeichnung EN hinzugefügt. EN steht für "Entfernen". Sollte also nach der Zuweisung eine Studierendenlösungsdatenstruktur einer EN Zeile zugewiesen werden, wird dem Studierenden empfohlen, diese Datenstruktur zu löschen, da sie den Musterlösungsstrukturen nicht genügend ähnlich ist.

6.2.2 Linear Assignment Problem

Basierend auf den erstellten Längen-Matrizen wird ein Linear Assignment[2] durchgeführt. Das Ziel dieses Assignments ist es, die Zeilen(M1...Mn) und Spalten(S1...Sn) so zu kombinieren, dass die Summe der durch die Kombination aus Spalte und Zeile gegebenen Zellen möglichst gering ist. Ein naiver Ansatz, dieses Problem zu lösen wäre es, alle möglichen Kombinationen auszuprobieren und die Kombinationen, welche zur kleinste gefundene Summe geführt haben, zurückzuliefern. Dies würde bei einer Matrix der Dimensionen n x n zur Überprüfung von n-Fakultät (n!) führen. Wenn das mit Zahlen dargestellt wird, ist dies für eine 3x3-Matrix mit $3! = 6$ eine relativ kleine Anzahl von Möglichkeiten. Bereits bei einer Matrix von 10x10 steigt die Anzahl Möglichkeiten auf $10! = 3628800$.
Der Hungarian algorithm[3] hat in der originalen Implementierung eine Laufzeitkomplexität von $O(n^4)$. Wendet man diesen Algorithmus auf das zuvor erwähnte Beispiel einer 10x10 Matrix an erhält man das Resultat $10^4 = 10000$, was um eine Vielzahl besser ist als 10!.
Als Grundlage für die Implementierung wurde der Algorithmus auf der Webseite von Noldorin[4] verwendet. Diese Grundlage verfügt jedoch nicht über die Möglichkeit mit Fliesskommazahlen zu arbeiten. Der Vergleich zweier Datenstrukturen kann aber einen Metrik-Wert zurückliefern, welcher nicht zwingend eine ganze Zahl ist. Aus diesem Grund wurde der Algorithmus von Noldorin so angepasst, dass keine Zahlenvergleiche mit == gemacht werden, sondern jeweils überprüft wird, ob die Ungleichheit

[2] http://en.wikipedia.org/wiki/Assignment_problem[9]
[3] http://en.wikipedia.org/wiki/Hungarian_algorithm[10]
[4] http://blog.noldorin.com/2009/09/hungarian-algorithm-in-csharp/[11]

zweier Zahlen in einem gewissen Bereich ist. Dies führte zu einer Version, welche ein Linear Assignment auf Matrizen mit Fliesskommazahlen durchführen kann.

	S1	S2	S3	S4
M1	1	4	2	6
M2	2	1	3	4
M3	6	7	3	1
M4	4	2	8	5

Tabelle 6.4: Gefundene Kombinationen des Hungarian algorithm

Der Hungarian algorithm liefert für die Tabelle 6.4 die Kombinationen [1,3], [2,1], [3,4] und [4,2]. Die dazugehörigen Zellen sind rot eingefärbt. Für die Tabellen 6.2 und 6.3 ist die Ausgabe dieselbe.

6.3 Lösungsprüfungsoptionen

In der vorhandenen Version des Lösungsprüfers wurde nur eine Lösungsprüfungsoption pro Aufgabe abgespeichert. Es handelt sich dabei um die Information, ob die Datenstrukturen auf konkrete Zahlenwerte oder auf das Vorhandensein einer geforderten Ordnung überprüft werden. Auch die Gewichte einer Aktionen (Hinzufügen eines Elementes, Swap, ...) wurden fest im Lösungsprüfer codiert und konnten vom Benutzer nicht mehr verändert werden. Auch die verfügbaren Aktionen wurden direkt im Lösungsprüfer definiert und konnten vom Entwickler einer Datenstruktur in Algoria nicht erweitert werden. Aus diesem Grund wurde dem *ISolutionCheckable*-Interface das Feld *ActionsAvailableList* hinzugefügt. Über dieses Feld kann *Algoria Worksheet* bei einer Datenstruktur abfragen, welche Aktionen diese unterstützt und welche Standardgewichtungen ein Entwickler vorsieht. Zum Beispiel ist das Hinzufügen eines Elementes in einem Array sehr viel teurer als in einer einfach verketteten Liste. Für die Umsetzung der *ActionsAvailableList* wurde die Klasse *Action* (siehe Kapitel 6.3.1) eingeführt und implementiert.

Die Administrationsversion von *Algoria Worksheet* wurde so erweitert, dass basierend auf den Datenstrukturen der Musterlösung die Dozierenden die Möglichkeit haben, das Gewicht einer Aktion selbst festzulegen. Die verfügbaren Aktionen können über das *ISolutionCheckable*-Interface abgefragt werden. Implementiert eine Datenstruktur dieses Interface nicht, wird dies dem Benutzer gemeldet (siehe Linked List in Abbildung 6.5).

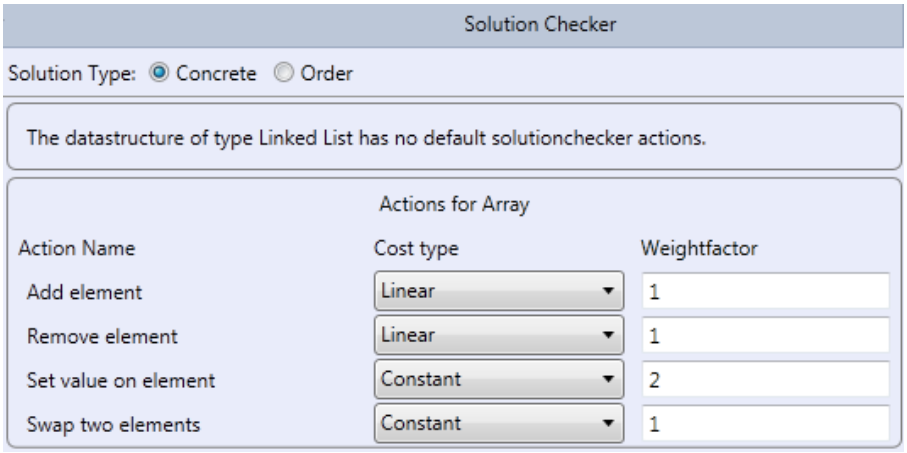

Abbildung 6.5: Lösungsprüfungsoptionen für ein Array und eine LinkedList

Für das Array in Abbildung 6.5 wurden vier Aktionen definiert. Es handelt sich um das Hinzufügen eines Elementes, das Löschen eines Elementes, dem Wert setzen auf einem Element und dem Swap

zweier Elemente. Diese Aktionen haben alle ein vorgegebenes Standardgewicht. Wie zu sehen ist, sind für die Gewichte nicht nur konstante Werte möglich, sondern auch solche, welche sich an der Länge der zu manipulierenden Datenstruktur orientieren. Die Werte, welche hier definiert werden, werden im Lösungsprüfer verwendet, um den Metrik-Wert zu berechnen. Dozierende haben so die Möglichkeit für jede Aufgabe andere Werte zu setzen, je nachdem was für die Aufgabe von zentraler Bedeutung ist.

6.3.1 Action

Action ist eine abstrakte Klasse, die eine Aktion auf einer Datenstruktur repräsentiert. Beispiele für solche Aktionen sind im vorhergehenden Kapitel zu finden. Im Codeausschnitt 6.2 ist das Grundgerüst der Klasse zu sehen. Mithilfe des *CostType* Feldes ist es möglich einer Aktion den Typ der Kostenberechnung zu setzten. Dieser Typ kann konstant, linear, logarithmisch oder quadratisch sein. Das Feld *WeightFactor* hält dann den vom Dozierenden gewählten Gewichtsfaktor. Mithilfe der *CalculateCost*-Methode, welche die Grösse der Datenstruktur verlangt, wird dann das Ausmass einer Aktion berechnet.

```
1  public abstract class Action
2  {
3      public CostType CostType { get; set; }
4      public abstract string Name { get; }
5      public float WeightFactor { get; set; }
6
7      public double CalculateCost(int datastructureSize)
8      {
9          switch (this.CostType)
10         {
11             case CostType.Constant:
12                 return this.WeightFactor;
13
14             case CostType.Linear:
15                 return this.WeightFactor * datastructureSize;
16
17             case CostType.Logarithmic:
18                 return this.WeightFactor * Math.Log(datastructureSize);
19
20             case CostType.Quadratical:
21                 return this.WeightFactor * Math.Pow(datastructureSize, 2);
22         }
23         return 0;
24     }
25 }
```

Codeausschnitt 6.2: Grundgerüst der Action-Klasse

Der Lösungsprüfer von *Algoria Worksheet* enthält drei Standardaktionen, welche von jeder Datenstruktur (sollte diese *ISolutionCheckable* implementieren) zur Verfügung gestellt werden müssen. Es handelt sich hierbei um *AddAction*, *RemoveAction* und *SetValueAction*. Diese Aktionen wurde schon implementiert und müssen von den Datenstrukturen nur in die *ActionAvailableList* aufgenommen werden. Wie sichergestellt wird, dass eine Klasse diese Aktionen zur Verfügung stellt, wird im Kapitel 7.1 behandelt.

6.4 Multiple Choice Lösungsprüfer

Nebst der automatischen Korrektur für Aufgaben, in welcher eine Datenstruktur gezeichnet werden muss, ist der Lösungsprüfer auch in der Lage, Multiple Choice Aufgaben zu korrigieren. Die Entscheidung, ob eine Aufgabe richtig oder falsch ist, ist sehr viel einfacher zu bewältigen. Im Vergleich zu den Datenstrukturen kann hier pro Aussage nur zwischen ganz richtig oder ganz falsch entschieden werden. Eine einzelne Aussage in einer Multiple Choice Aufgabe kann auch nur diese 2 Zustände haben. Eine ganze Aufgabe jedoch kann zu einem gewissen Prozentsatz richtig oder falsch sein. Sind zum Beispiel drei von vier Aussagen korrekt markiert, resultiert dies in einer 75% richtigen Lösung. Damit die Studierenden auch hier eine unmittelbare Rückmeldung über ihre Leistungen haben, werden die Aussagen eingefärbt. Ein roter Hintergrund bedeutet, dass die Aussage nicht korrekt markiert wurde. Ein grüner Hintergrund resultiert aus einer korrekten Antwort. Als Beispiel für diese Rückmeldung ist Abbildung 6.6 zu betrachten.

Abbildung 6.6: Rückmeldung des Lösungsprüfers für Multiple Choice

6.5 Visualisierung der Metrik-Werte

Damit die Ähnlichkeit einer Datenstruktur in der Studierendenlösung zu ihrem Gegenüber in der Musterlösung gemessen werden kann, wird im Lösungsprüfer eine Metrik verwendet. Diese ist für die Berechnungen im Linear Assignment ideal. Für eine aussagekräftige Rückmeldung dem Studierenden gegenüber ist dies nicht praktikabel. Aus diesem Grund wurden den Überprüfungsmethoden im *ISolutionCheckable*-Interface ein zusätzliches Feld im Rückgabetupel gegeben, welches die schlechtes mögliche Metrik angibt (siehe 6.1). Mithilfe dieses zweiten Wertes kann berechnet werden, zu wie viel Prozent sich die beiden Datenstrukturen ähneln. Im Beispiel in der Abbildung 6.7 ist zu sehen, dass ein Übereinstimmungsgrad von 92% erreicht wurde. Da scheinbar einige Werte aus der Studierendenlösung in der Musterlösung nicht vorhanden sind, wurden 100% nicht erreicht. Zusätzlich gibt der Lösungsprüfer die Rückmeldung, dass die Datenstruktur des Typs “Linked List” in der Musterlösung nicht vorkommt und deshalb gelöscht werden sollte.

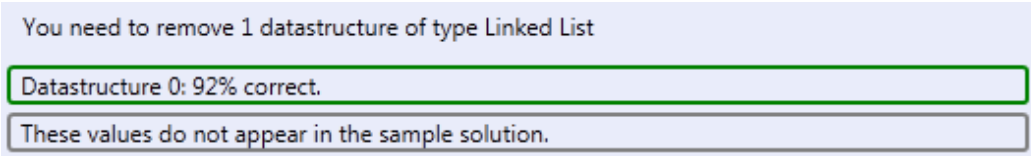

Abbildung 6.7: Rückmeldung des Lösungsprüfers für Datenstrukturen

7 Code Contracts

Validierung innerhalb des Codes ist eine Aufgabe, welche in jedem Programm notwendig ist. Es gibt verschiedene Möglichkeiten zum Beispiel den Input einer Methode zu validieren. Eine von zwei bekannten Möglichkeiten ist die Verwendung von If Blöcken, welche zum Werfen einer Exception führen, sollte eine Bedingung verletzt werden. Eine andere Möglichkeit ist die Verwendung von Assertions, die bei der Verletzung einer Bedingung ein Fehler melden. Die beiden folgenden Codeausschnitte zeigen eine Anwendung dieser Konzepte für eine Methode, welche zwei Zahlen addiert. Voraussetzung dafür ist, dass keine der beiden Zahlen kleiner als 0 ist.

```
1 public int AddNumbers(int number1, int number2)
2 {
3     if (number1 < 0) throw new NumberNotPositiveException("Number1");
4     if (number2 < 0) throw new NumberNotPositiveException("Number2");
5
6     return number1 + number2;
7 }
```

Codeausschnitt 7.1: Validierung mittels If Blöcken

```
1 public int AddNumbers(int number1, int number2)
2 {
3     Debug.Assert(number1 >= 0, "number1 is negative");
4     Debug.Assert(number2 >= 0, "number2 is negative");
5
6     return number1 + number2;
7 }
```

Codeausschnitt 7.2: Validierung mittels Assertions

Zusätzlich zu diesen bekannten Validierungsstrategien wurde von den Microsoft Research Labs ein neues Konzept eingeführt, Code Contracts. Das folgende Beispiel zeigt, wie dieselben Validierungen mit Code Contracts gelöst werden können.

```
1 public int AddNumbers(int number1, int number2)
2 {
3     Contract.Requires(number1 > 0, "Number1 is negative");
4     Contract.Requires<NumberNotPositiveException>(
5         number2 > 0, "Number1 is negative");
6     Contract.Ensures(Contract.Result<int>() > 0, "Result is negative");
7
8     return number1 + number2;
9 }
```

Codeausschnitt 7.3: Validierung mittels Code Contracts

Der Codeausschnitt 7.3 zeigt nicht nur die Inputvalidierung sondern auch die Validierung des Resultates. Zu sehen ist, dass alle Bedingungen, an einem zentralen Ort zu finden sind. Würde mit einem If-Block oder Assertions validiert werden, müssten diese Validierungen vor dem Return-Statement stehen, was das Auffinden auf den ersten Blick erschwert. Auch die Möglichkeit eine Exception zu werfen ist gegeben. Dazu wird der Typ der zu werfenden Exception in Klammern vor dem zu validierenden Ausdruck angegeben (siehe Zeile 4 in Codeausschnitt 7.3).

Neben Input- und Outputvalidierung liefern Code Contracts viele zusätzliche Möglichkeiten Validierungen durchzuführen. Eine davon ist die Überprüfung einer Invarianten eines Objektes, welche immer gegeben sein muss. Beispielsweise hat ein Konto-Objekt einer Band die Invariante, dass der Kontostand immer positiv sein musst. Dies kann durch eine einzelne Zeile in einer als "ContractInvariantMethod" annotierten Methode (zu sehen im Codeausschnitt 7.4) erreicht werden.

```
1 [ContractInvariantMethod]
2 private void ObjectInvariant()
3 {
4     Contract.Invariant(AccountBalance > 0);
5 }
```

Codeausschnitt 7.4: Definition einer Invarianten mit Code Contracts

Die Durchsetzung von Vorbedingungen über Interfaces oder abstrakte Klassen wird auch durch Code Contracts gegeben. Eine Anwendung dieses Konzeptes ist auch in *Algoria Worksheet* zu finden. Dies wird in Kapitel 7.1 beschrieben. Dadurch, dass die Code Contracts direkt von Microsoft entwickelt wurden, ist auch die Integration in Visual Studio gut umgesetzt. Dies ermöglicht, dass die Resultate des statischen Validierens (was den Code zur Kompilierungszeit überprüft) und die zur Laufzeit gefundene Verletzungen von Bedingungen als Warnung an Visual Studio geliefert werden können.

7.1 Einbindung in Algoria Worksheet

Die Verwendung von Code Contracts zur Validierung von Vor- und Nachbedingungen wurde bereits im vorhergehenden Kapitel beschrieben und fand in *Algoria Worksheet* die beschriebene Anwendung. Auch das Durchsetzen von Contracts durch ein Interface wurde im Lösungsprüfer von *Algoria Worksheet* verwendet und soll in diesem Kapitel erläutert werden.

Für das Definieren und Durchsetzen von Code Contracts für und durch Interfaces benötigt es immer das Interface selbst und eine Standardimplementierung (die Contract Klasse), in welchen die Code Contracts definiert werden. Im Folgenden wird das Interface *ISolutionCheckable* verwendet. Die Bedingung hier ist, dass jede Klasse, welche das Interface implementiert, in der *ActionsAvailableList* mindestens die drei Standardaktionen zur Verfügung stellen muss. Dies sind die *AddAction*, die *RemoveAction* und die *SetValueAction*. Auch dass einer Check-Methode in einem Interface keine Null-Pointer übergeben werden dürfen, und wie der Rückgabewert auszusehen hat, wird in den folgenden Codeausschnitten dargestellt.

```
1 [ContractClass(typeof(SolutionCheckableContracts))]
2 public interface ISolutionCheckable
3 {
4     IEnumerable<Action> ActionsAvailableList { get; }
5     Tuple<double, double, IEnumerable<Hint>> CheckForConcreteNumbers(...);
6     Tuple<double, double, IEnumerable<Hint>> CheckForOrder(...);
7 }
```

Codeausschnitt 7.5: Contract Klasse für ein Interface definieren

Der Codeausschnitt 7.5 zeigt noch einmal das Interface *ISolutionCheckable*. Zusätzlich wurde die Klassen-Annotation "[ContractClass(typeof(SolutionCheckableContracts))]" eingefügt. Diese bewirkt, dass in jeder Klasse, welche das Interface implementiert, automatisch die in der Contract-Klasse definierten Bedingungen übernommen werden. In diesem Falle sollen die Bedingungen in der *SolutionCheckableContracts*-Klasse gesucht werden.

Nachfolgend zeigt der Codeausschnitt 7.6 die Contract-Klasse für das *ISolutionCheckable* Interface. Auch diese Klasse verfügt, analog zum Interface, über eine Anntoation. In dieser wird angegeben, für welches Interface es sich um die Contract-Klasse handelt.

```
1  [ContractClassFor(typeof(ISolutionCheckable))]
2  internal sealed class SolutionCheckableContracts : ISolutionCheckable
3  {
4      public IEnumerable<Action> ActionsAvailableList
5      {
6          get
7          {
8              Contract.Ensures(Contract.Result<ReadOnlyCollection<Action>>().
9                  Any(x => x is AddAction));
10             ...
11             return default(ReadOnlyCollection<Action>);
12         }
13     }
14
15     public Tuple<double, double, IEnumerable<Hint>> CheckForConcreteNumbers(
16         IDatastructure<DatastructureElement> sampleDs,
17         DatastructureHandle studentDs, Metric metric)
18     {
19         Contract.Requires(
20             sampleDs != null &&
21             studentDs != null &&
22             studentDs.Datastructure != null);
23         Contract.Ensures(
24             Contract.Result<Tuple<int, IEnumerable<Hint>>>().Item1 >= 0 &&
25             Contract.Result<Tuple<int, IEnumerable<Hint>>>().Item2 != null);
26         return default(Tuple<double, double, IEnumerable<Hint>>);
27     }
28
29     public Tuple<double, double, IEnumerable<Hint>> CheckForOrder(
30         IDatastructure<DatastructureElement> sampleDs,
31         DatastructureHandle studentDs, Metric metric)
32     {
33     ...
34     }
35 }
```

Codeausschnitt 7.6: Contract Klasse für ein Interface

Durch die Implementierung des gegebenen Interfaces, muss auch die Contract-Klasse über ein *ActionsAvailableList* Feld verfügen. Innerhalb der get-Methode des Properties können nun die verlangten Tests durchgeführt werden. Da es sich um eine Outputvalidierung handelt, wird ein *Contract.Ensures*-Aufruf verwendet, welcher im Resultat nach einem Objekt sucht, welches vom Typ *AddAction* ist. Dieser Aufruf kann analog für die restlichen Standardaktionen verwendet werden. Anzumerken ist die Verwendung der *Contract.Result¡¿*-Methode. Diese gibt eine Referenz auf das zu returnierende Objekt zurück. In den eckigen Klammern kann dazu der Typ des Objekts definiert werden. Zum Abschluss der Methode wird ein "default"-Objekt vom geforderten Typ erstellt und zurückgegeben.

Zusätzlich zu dieser Validierung des Rückgabewertes wird durch die Contract-Klasse auch eine Überprüfung von Eingabewerten der Check-Methoden forciert. Dies geschieht analog zum bereits gezeigten Beispiel im Codeausschnitt 7.3.

8 Kollaboration

Während den weiteren Projektarbeiten 8 und 9 wird das Programm *Algoria Worksheet* um die Funktionalität der Kollaboration erweitert. Im folgenden Kapitel wird das Konzept zur Kollaboration beschrieben und Änderungen an den Programmen geplant und veranschaulicht. In einem ersten Unterkapitel werden bereits bestehende Lösungen im Bereich Lernsoftware und der elektronischen Kollaboration im Allgemeinen behandelt und erklärt, warum diese von Bedeutung sind. Die restlichen Unterkapitel befassen sich mit der Kollaboration in Algoria und den damit auftauchenden Schwierigkeiten.

8.1 Educational Software

Dieses Unterkapitel stellt bestehende Lösungen auf dem Softwaremark im Bereich der Lernsoftware vor. Interessant sind auch Produkte, welche bereits etwas Kollaboration miteinbeziehen oder solche, die eine vergleichbare Art der Lösungsüberprüfung haben, wie sie in *Algoria Worksheet* vorhanden ist. Wenn es solche Software geben sollte, könnte *Algoria Worksheet* sich in einem bestehenden System weiterentwickeln ohne die gesamte Kollaborationsthematik selbst gelöst zu haben.

Zu Beginn soll hier der Bereich der Lernsoftware erfasst und interessante Produkte daraus vorgestellt werden. Lernsoftware umfasst hauptsächlich das Erlernen des Zehnfinger Tippsystems am Computer, das Aneignen von Rechtschreibregeln und das Üben von Grundrechenoperationen. Die Zielgruppen der allgemeinen Lernsoftware sind hauptsächlich Kindern. Eine Ausnahme sind Programme, welche beim Erlernen einer Sprache helfen sollen. Im Informatik- und Softwareentwicklungsbereich ist wenig Lernsoftware zu finden. Nebst dem alleinigen Lernen mit der Unterstützung eines Computerprogrammes hat auch die Kollaboration in den Schulzimmern ihren Platz gefunden. Die Kollaboration im Unterricht wird jedoch häufig mit der Verwendung eines interaktiven Whiteboards[1] in Verbindung gebracht. Auch der Einsatz eines *interactive response system*[2] zur Übermittlung von Lösungen vom Studierenden zum Dozierenden ist eine Möglichkeit. Gemeinsames Bearbeiten von Aufgaben unter Studierenden ist ein kaum vertretenes Teilgebiet in Educational Software.

Bei den Softwareanbietern von Lernsoftware mit Informatikbezug sind *LJ CREATE*[3], welche in mehreren Bereichen der Informatik Lernmodule anbietet, und *Cisco*[4], die für den Netzwerkbereich Übungssoftware entwickelt, zu nennen. Als generelle Lernsoftware ohne eine direkte Verbindung zur Informatik wird das Kursmanagementsystem *Moodle*[5] betrachtet.

Die Palette von *LJ CREATE* umfasst sowohl das Modul *Network Technology*[6] als auch das Programm *Computer Programming*[7]. Das hier zweit genannte *Computer Programming* ist vom Themengebiet her *Algoria Worksheet* am ähnlichsten. Es befasst sich mit den Grundlagen der Programmierung. Übungen, in welchen kleine Codestellen geändert werden müssen um die Auswirkung auf das gesamte Programm zu testen, werden ebenfalls angeboten.

Der *Packet Tracer*[8] von *Cisco* bietet eine Simulationsumgebung für das Zusammenstellen von Netzwerken und dem Konfigurieren von Routern, Computern und Switches. In diesem Programm kann ein Administrator eine Übung für Studierende erstellen, welche danach von Studierenden Schritt für

[1] http://www.smarttech.com/us/Solutions/Education+Solutions/Products+for+education/Interactive+whiteboards+and+displays/SMART+Board+interactive+whiteboards[12]
[2] http://www.smarttech.com/us/Solutions/Education+Solutions/Products+for+education/Complementary+hardware+products/SMART+Response[13]
[3] http://www.ljcreate.com/[14]
[4] http://www.cisco.com[15]
[5] http://www.moodle.org[16]
[6] http://www.ljcreate.com/products/product.asp?id=180&program=183&curr=13[17]
[7] http://www.ljcreate.com/products/product.asp?id=44&program=183&curr=13[18]
[8] http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html[19]

Schritt abgearbeitet werden kann. Mit einer Prozentangabe können die Studierenden überprüfen, wie weit sie schon fortgeschritten sind und wie nahe am Ziel sie sich bereits befinden.

Moodle bietet zum Einen die Möglichkeit Daten über eine Datenablage auszutauschen und zum Anderen das Erstellen von Übungen sowie Prüfungen. Die Datenablage kann nicht nur dafür genutzt werden, den Studierenden Informationen zukommen zu lassen, sondern ermöglicht auch eine Abgabe von Arbeiten über eine Upload-Funktion. Wen Übungen oder Prüfungen automatisiert korrigiert werden sollen, dürfen diese nur Multiple Choice Aufgaben beinhalten. Jeder sonstige Aufgabentyp bedarf einer Lehrperson, welche die Aufgabe manuell bewertet.

8.2 Algoria Worksheet Collaboration

Das Konzept für die Kollaboration in *Algoria Worksheet* dient als Vorlage für die Implementierung in einem Folgeprojekt. Zusätzlich stellt das Konzept sicher, dass während der kommenden Weiterentwicklung die Kollaboration nicht durch zusätzliche Erweiterungen beeinträchtigt wird. Die zentrale Frage, welche das Konzept beantworten soll, ist: „Wie arbeiten verschiedene Benutzer gleichzeitig an einer in *Algoria Worksheet* gestellten Aufgabe?“

Die folgenden Unterkapitel beinhalten eine Beschreibung der Anwendungsszenarien und der Änderungen an den bereits bestehenden Versionen von *Algoria Worksheet*. Letzteres beinhaltet auch Softwarepakete, welche neu zu erstellen sind.

8.2.1 Szenarien

Die folgenden vier Szenarien veranschaulichen, wie das Arbeiten mithilfe der Kollaboration in *Algoria Worksheet* abläuft. Es werden im Folgenden vier denkbare Ausgangslagen präsentiert. Jedes Szenario beschreibt auch, wie sich die einzelnen Komponenten (bereits bestehende, sowie neu zu entwickelnde) an der Kollaboration beteiligen.

Gemeinsames Arbeiten Offline

Zwei Studierende oder mehrere arbeiten gemeinsam an einer gestellten Aufgabe. Die Studierenden sind sich räumlich nahe. *Algoria Worksheet* liefert für dieses Anforderung eine Zeichenfläche, auf welcher mehrere Benutzer gleichzeitig arbeiten können. Diese wird von einem Mini-Server bereitgestellt, welcher auf dem Rechner gestartet wird, auf welchen sich die restlichen Studierenden verbinden. Die Synchronisation der Modifikationen auf der Zeichenfläche wird ebenfalls von diesem Mini-Server übernommen. Es ist denkbar, dass dieses Szenario nicht nur in einem Netzwerk mit Ethernet oder Wireless Lan Anwendung findet. Auch ein Arbeiten mit Peer-to-Peer Methoden wie zum Beispiel die Kommunikation zwischen 2 Benutzern via Infrarot ist denkbar.

Gemeinsames Arbeiten Online

Zwei oder mehrere Studierende arbeiten gemeinsam an einer gestellten Aufgabe. In diesem Szenario spielt die räumliche Distanz keine Rolle. Da zwei Studierende nicht zwingend in einem gemeinsamen Netzwerk, sondern sich zuhause befinden können und miteinander arbeiten wollen, ist eine direktes Herstellen einer Verbindung nicht möglich. Hier bietet sich eine Verbindung über das Internet an. Für diese Aufgabe ist ein Serverprogramm vorgesehen, welches die Verbindung zwischen den einzelnen *Algoria Worksheet* Instanzen verwaltet. Zusätzlich erlaubt eine Serverinstanz das Bereitstellen von Aufgaben und Lösungen vonseiten eines Dozierenden oder von Studierenden.

Arbeiten während des Unterrichts

Der Dozierende stellt während des Algorithmen und Datenstrukturen Unterrichts Aufgaben, welche von den Studierenden bearbeitet werden sollen. Die Studierenden können mit *Algoria Worksheet* auf eine Server Applikation zugreifen und eine Aufgabe herunterladen. Während der lokalen Bearbeitung können Teillösungen auf den Server geladen werden, welche vom Dozierenden heruntergeladen und angezeigt werden können. Der Vorgang des Hochladens von Studierendenlösungen, als auch das Herunterladen auf Dozierenden-Seite kann automatisiert werden. So wird eine Art "Live-Übertragung der Aufgabenbearbeitung" ermöglicht. In diesem Szenario handelt es sich um eine Abwandlung des vorhergehenden Modelles. Der Dozierende arbeitet gewissermassen mit einem Studierenden zusammen, um so immer Zugriff auf die Lösung zu haben.

Diese "Live-Übertragung der Aufgabenbearbeitung" muss nicht zwingend auf dem Notebookdisplay des Dozierenden dargestellt werden. Eine denkbar Erweiterung dieses Szenarios ist die Verwendung eines sehr grossen Displays, welches als Wandtafelersatz verwendet wird. Die Oberfläche von *Algoria Worksheet* müsste für diese Anforderung minimal angepasst werden. Es könnte, analog zu den bisherigen Algoria Engine Reitern, ein Reiter implementiert werden, welcher über mehrere Algoria Zeichenflächen verfügt. In jeder dieser Engines kann der Bearbeitungsstand eines Studierenden angezeigt werden. Durch die grosse zur Verfügung stehende Fläche eines Giga-Displays können wesentlich mehr Zeichenflächen nebeneinander dargestellt werden als auf einem Standarddisplay oder einem Beamer. Innerhalb eines solchen Reiters könnten Andockstellen definiert werden, sodass die Zeichenflächen beliebig angeordnet werden können und der Dozierende zwei verschiedene Lösungen gleichzeitig aufzeigen und besprechen kann.

Erarbeiten eines Testates

Aufgaben können nicht nur während der Unterrichtslektion gestellt werden, sondern auch als Testat-Aufgabe von den Studierenden bearbeitet werden. Diese sollen z.B. in Heimarbeit bearbeitet und danach abgegeben werden. Die Aufgaben werden auf der Serverapplikation zur Verfügung gestellt. Studierende können sie hier herunterladen, bearbeiten und die Lösung auf diesem Wege abgeben. Die Dozierenden können die fertigen Lösungen herunterladen und bewerten. Eine gemeinsame Zeichenfläche wird in diesem Szenario benötigt.

8.2.2 Erweiterungen an Algoria und Algoria Worksheet

Die Erweiterungen an *Algoria* und *Algoria Worksheet* sowie die neu zu erstellende Software, welche sich aus den Szenarien herauskristallisierten, werden im Folgenden beschrieben.

Gemeinsame Zeichenfläche

Damit mehrere Studierende an einer Aufgabe arbeiten und die Datenstrukturen simultan verändern können, müssen Änderungen an den Zeichenflächen übertragen und synchronisiert werden. Dies führt zu zusätzlichen Aufgaben für *Algoria*. Diese sind:

- Senden von Veränderungen der lokalen Zeichenfläche
- Empfangen von Änderungen an den Zeichenflächen der restlichen Benutzer
- Synchronisieren der Änderungen

Diese zusätzlichen Aufgaben können von *Algoria* im momentanen Zustand nicht übernommen werden. Deshalb bietet sich das Erweitern von *Algoria Worksheet* mit einem Server- beziehungsweise einem Client-Modul an. Dieses Modul übernimmt das Versenden, Empfangen und Synchronisieren

von Updates der Zeichenfläche.

Serverapplikation

Eine weitere zu erstellende Komponente für die Kollaboration ist eine Serveranwendung. Der Server übernimmt folgende Aufgaben:

- Herstellen von Verbindungen zwischen Benutzern, welche nicht in demselben Netz sind
- Verwalten von Aufgaben für Algoria Worksheet
  - Upload von Aufgaben (Dozierende)
  - Download von Aufgaben (Studierende)
  - Upload von Lösungen (Studierende)
  - Download von Lösungen (Dozierende)

8.2.3 Updates in Kollaboration

Die Kommunikation unter den verschiedenen *Algoria Worksheet*-Instanzen, die gleichzeitig an einer Aufgabe beteiligt sind, bedarf einer etwas genaueren Betrachtung. So stehen verschiedene Möglichkeiten zur Verfügung, was genau in einem Update übertragen wird. Die folgenden Möglichkeiten sind denkbar:

- Jeder Strich auf der Zeichenfläche
- Jeder Strich, welcher von Algoria erkannt wurde
- Jede erkannte Veränderung an einer Datenstruktur, bzw. das Hinzufügen einer neuen Datenstruktur

Je früher die Änderung übertragen wird, desto einfacher ist die Synchronisation. Wird ein Strich übertragen, welcher in Algoria noch nicht prozessiert wurde, können sich zuerst die beteiligten *Algoria Engine*-Instanzen darauf einigen, ob der Strich durch einen Ausschluss verunmöglicht wird, bevor er an die Algoria Engines weitergeleitet wird. Ein solcher Ausschluss wäre möglich, wenn eine andere Instanz an derselben Stelle bereits ein Strich versendet hat, welcher grössere Priorität hat, z.B. eine Datenstruktur vervollständigt. Werden nur Veränderungen an bereits erkannten Datenstrukturen übertragen, erhöht sich die Komplexität der Synchronisation schnell. Die zentrale Frage, die angegangen werden müsste, ist das Vorgehen, wenn gleichzeitig in zwei Instanzen an derselben Stelle eine Datenstruktur fertiggestellt wird.

9 Lösungsprüfer Testmodus

Nebst dem Konzept für die Kollaboration wurde während der Projektarbeit zweites, wesentlich weniger umfangreiches Konzept entworfen. Es handelt sich dabei um die Prüfung des Lösungstesters. Im Zentrum steht die Frage:"Wie ändern sich die Metrik-Werte für eine gegebene Studierendendatenstruktur, welche mit einer gegebenen Musterlösungsdatenstruktur verglichen wird, wenn verschiedene Lösungsprüfungsoptionen verwendet werden?".

Für die Beantwortung dieser Frage ist es notwendig, zwei Datenstrukturen automatisch und unter Berücksichtigung von verschiedenen Einstellungen des Lösungsprüfers zu vergleichen. Da schon viele Elemente, welche bei der Lösung eines solchen Problems benötigt werden, in *Algoria Worksheet* vorhanden sind, macht es Sinn, diese zu verwenden und möglichst wenig neue zu entwickeln.

Eine denkbare Umsetzung der automatischen Prüfung des Lösungsprüfers sieht vor mit der schon vorhandenen Administrationsversion zu arbeiten. Der Ablauf ist wie folgt:

- Eröffnen eines neuen Arbeitsblattes
- In der leeren Aufgabe 1 eine Musterlösung eintragen
- In der leeren Aufgabe 1 eine Studierendenlösung eintragen
- In einer Konfigurationskonsole die zu testenden Lösungsprüfungsoptionen eintragen
- Den Test ausführen
- Die Testresultate werden in einem Fenster gezeigt oder auf Wunsch in eine Datei gespeichert

In diesem Vorgang würde es nur zwei neue Elemente benötigen. Dies ist die Konfigurationskonsole, welche es ermöglicht, ein Reihe von Lösungsprüfungsoptionen zu erstellen und das Fenster zur Darstellung der Resultate.

Weiter wäre es denkbar, einen Testmodus der *Algoria Worksheet* Administrationsversion anzubieten, in welcher immer nur eine Aufgabe vorhanden ist, aber das direkte Ausführen des Lösungsprüfers möglich ist, sodass Dozierende alternative Musterlösungen miteinander abgleichen können.

10 Reflexion

In den folgenden Unterkapiteln wird sowohl das vergangene Projekt in den Bereichen Projektmanagements, Softwareentwicklung und Konzeptgestaltung reflektiert, als auch ein Ausblick auf zukünftige Projekte gegeben. Das Kapitel 10.1 beinhaltet sowohl die Punkte der Planung als auch die der allgemeinen Projektdurchführung. Im nachfolgenden Kapitel werden Stärken und Schwächen während der Software- und Konzeptentwicklung zusammengetragen. Abschliessend bietet Kapitel 10.3 einen Ausblick auf folgende Projekte.

10.1 Projektmanagement

Da das Projekt nicht von einem Team sondern von einer Einzelperson durchgeführt wurde, spielt die Gruppendynamik während des Projektes keine Rolle. Von einer grossen Bedeutung war aber die Beziehung zwischen Advisor und Studierenden. Durch wöchentliche Sitzungen wurde ein stetiger Dialog ermöglicht, welcher sich positiv auf den Ausgang des Projektes ausgewirkt hat. Dies sollte unbedingt beibehalten werden.
Die Planung wurde zu Beginn des Semesters erstellt und mit einer Projektklärung unterschrieben. Dieses Vorgehen hat viele Vorteile und ist auch weiterhin beizubehalten. Eine Verbesserung beim Umsetzen des Plans selbst ist jedoch die Aufteilung der Dokumentation in kleine Teile, welche jeweils nach dem Abschliessen einer Projektphase platziert wurden. Diese wurden grösstenteils nicht eingehalten. Ein Schreiben der Dokumentation während des Semester sollte noch einmal ins Auge gefasst werden, die Teile währen des Semesters sollten aber kleiner sein. Auch sollte der Beginn der Dokumentationsphase strikt eingehalten werden und allfällige kleine Erweiterungen und das Beseitigen von kleinen Fehlern erst nach einem gewissen Fortschritt in der Dokumentation angegangen werden.
Eine gute Angewohnheit, welche beibehalten werden sollte, ist das Erfassen von geleisteten Arbeiten. Dies nicht nur, da eine Auswertung der Soll- und Ist-Stunden möglich ist, sondern auch weil gleichzeitig mit dem Erfassen Notizen für die abschliessende Dokumentation gemacht werden können.

10.2 Algoria Worksheet

Die Weiterentwicklung an *Algoria Worksheet* wurde begünstigt durch ein hohes Wissen, welches von der Bachelor Thesis kam, während welcher das Programm entstanden ist. Grundsätzlich wurden keine grösseren Probleme angetroffen. Die Einbindung von Code Contracts und das Einlesen in diese neue Thematik war eine spannende Erfahrung. Das Definieren von Style-Guides für die Code-Dokumentation forcierten einen Standard, welcher vorgab, wo eine XML Dokumentation vorhanden sein muss. Diese Style-Guides sollten während der folgenden Projekte weiterverwendet werden um so von Beginn an eine saubere Dokumentation des Codes zu haben und zu verhindern, dass diese kurz vor Projektabgabe geschrieben werden.
Das Erstellen eines Konzeptes für die Kollaboration und den automatisierten Lösungsprüfungstest waren anspruchsvollere Arbeiten als zu Beginn gedacht. Durch das stetige Besprechen des Konzeptes für die Kollaboration während den wöchentlichen Sitzungen konnte dies aber über eine gewisse Zeit reifen, bevor es den Weg auf das Papier fand.

10.3 Ausblick

In den folgenden zwei Semestern wird *Algoria Worksheet* kontinuierlich weiterentwickelt. Einerseits auf Grundlage der hier erarbeiteten Konzepte, andererseits ist es möglich, dass Benutzer, welche sich mit

dem Programm beschäftigen, Wünsche und Anregungen haben. Die Projektarbeit 8 könnte sich mit den Grundlagen für die "Live-Übertragung" beschäftigen. Dies würde bedeuten, dass ein Container erstellt wird, welcher für den Einsatz auf einem sehr grossen Display geeignet ist, und Platz für mehrere, frei platzierbare Algoria Zeichenflächen bietet. Zudem sollte die Thematik des Synchronisierens von Zeichenflächen weiterverfolgt und bereits bestehende Lösungen untersucht werden.

11 Ehrlichkeitserklärung

Hiermit bestätigt der Autor, diese Arbeit ohne fremde Hilfe und unter Einhaltung der gebotenen Regeln erstellt zu haben.

Reto Frey

Ort, Datum Unterschrift

Literaturverzeichnis

[1] Aufgabenstellung Algoria - Lernumgebung
Christoph Stamm (2011)

[2] Code Contracts [online]
http://research.microsoft.com/en-us/projects/contracts/ 2012.

[3] Aufgabenstellung Dozierende Usability Test
Reto Frey (2011)

[4] Microsoft Homepage zu Application Commands [online]
http://msdn.microsoft.com/en-us/library/system.windows.input.applicationcommands.aspx 2012.

[5] Microsoft Homepage zu WPF [online]
http://windowsclient.net/wpf/ 2012.

[6] Aufgabenstellung Studierende Usability Test
Reto Frey (2011)

[7] Avalon Dock Webseite [online]
http://avalondock.codeplex.com/ 2012.

[8] Microsoft Homepage zu MVVM [online]
http://msdn.microsoft.com/en-us/magazine/dd419663.aspx 2012.

[9] Wikipedia Artikel zu Assigment Problem [online]
http://en.wikipedia.org/wiki/Assignment_problem 2012.

[10] Wikipedia Artikel zum Hungarian algorithm [online]
http://en.wikipedia.org/wiki/Hungarian_algorithm 2012.

[11] Noldorin's Blog [online]
http://blog.noldorin.com/2009/09/hungarian-algorithm-in-csharp/ 2012.

[12] SMART Board interactive whiteboards [online]
http://www.smarttech.com/us/Solutions/Education+Solutions/Products+for+education/Interactive+whiteboards+and+displays/SMART+Board+interactive+whiteboards 2011.

[13] SMART Response interactive response systems [online]
http://www.smarttech.com/us/Solutions/Education+Solutions/Products+for+education/Complementary+hardware+products/SMART+Response 2011.

[14] Webseite der Firma *LJ CREATE* [online]
http://www.ljcreate.com/ 2011.

[15] Webseite der Firma *Cisco* [online]
http://www.cisco.com/ 2011.

[16] Webseite des Kursmanagementsystems *Moodle* [online]
http://www.moodle.org 2011.

[17] Webseite des *Network Technology* Moduls [online]
http://www.ljcreate.com/products/product.asp?id=180&program=183&curr=13 2011.

[18] Webseite des *Computer Programming* Lernprogramms [online]
http://www.ljcreate.com/products/product.asp?id=44&program=183&curr=13 2011.

[19] Webseite des *Packet Tracer* Lernprogramms [online]
http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html
2011.

Abbildungsverzeichnis

Tabellenverzeichnis

Codeausschnitte